

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

Кафедра інженерії програмного забезпечення

КУРСОВА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

з дисципліни «Бази даних»

на тему:

**«Розробка сервісу опитування з використанням Bot API від
месенджера "Telegram"»**

студента II курсу групи ПЗк-21-1
спеціальності 121 «Інженерія
програмного забезпечення»
Каліновського Дениса Юрійовича
(прізвище, ім'я та по-батькові)

Керівник ст. викладач кафедри ПЗ
Олексій ЧИЖМОТРЯ
Дата захисту: "___" _____ липня 2022 р.
Національна шкала _____
Кількість балів: _____
Оцінка: ECTS _____

Члени комісії

_____	<u>Інна СУГОНЯК</u>
(підпис)	(ім'я та прізвище)
_____	<u>Ольга КОРОТУН</u>
(підпис)	(ім'я та прізвище)
_____	<u>Світлана КРАВЧЕНКО</u>
(підпис)	(ім'я та прізвище)
_____	<u>Олексій ЧИЖМОТРЯ</u>
(підпис)	(ім'я та прізвище)

Житомир – 2022
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
Факультет інформаційно-комп'ютерних технологій
Кафедра інженерії програмного забезпечення
Освітній рівень: бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»

«ЗАТВЕРДЖУЮ»
В. о. завідувача кафедри ІПЗ
_____ Андрій МОРОЗОВ
“ _____ ” _____ 2022р.

ЗАВДАННЯ
НА КУРСОВУ РОБОТУ СТУДЕНТУ
Каліновському Денису Юрійовичу

1. Тема роботи: Розробка сервісу опитування з використанням Bot API від месенджера “Telegram”.
2. Керівник курсової роботи: Олексій ЧИЖМОТРЯ
3. Строк подання студентом: “ _____ ” липня 2022р.
4. Вхідні дані до роботи: розробити базу даних сервісу опитування та додаток-бот на основі Bot API від “Telegram”.
5. Зміст розрахунково-пояснювальної записки(перелік питань. Які підлягають розробці)
 1. Постановка завдання.
 2. Аналіз аналогічних розробок.
 3. Алгоритми роботи програми.
 4. Опис роботи програми.
 5. Програмне дослідження.
6. Перелік графічного матеріалу(з точним зазначенням обов'язкових креслень)
 1. Презентація до КП
 2. Посилання на репозиторій: <https://gitlab.com/dk000/quizzzy>
7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посади консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3,4	Олексій ЧИЖМОТРЯ, ст. викладач кафедри ІПЗ	20.02.2022	20.02.2022

8. Дата видачі завдання “ 20 ” лютого 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів курсової роботи	Строк виконання етапів роботи	Примітки
1	Постановка задачі	20.02-25.02	виконано
2	Пошук, огляд та аналіз аналогічних розробок	26.02-15.04	виконано
3	Формулювання технічного завдання	16.04-20.04	виконано
4	Опрацювання літературних джерел	21.04-28.04	виконано
5	Проектування структури	01.05-15.05	виконано
6	Написання програмного коду	16.05-01.06	виконано
7	Відлагодження	02.06-10.06	виконано
8	Написання пояснювальної записки	13.06-20.06	виконано
9	Захист		

Студент

(підпис)

Денис КАЛІНОВСЬКИЙ

(прізвище та ініціали)

Керівник роботи

(підпис)

Олексій ЧИЖМОТРЯ

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до курсової роботи на тему «Розробка сервісу опитування з використанням Bot API від месенджера "Telegram"» складається з переліку умовних скорочень, вступу, чотирьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на 38 сторінках друкованого тексту.

Пояснювальна записка має 20 сторінок додатків. Список використаних джерел містить 10 найменувань і займає 1 сторінку. В роботі наведено 37 рисунків. Загальний обсяг роботи – 62 сторінки.

Ключові слова: БАЗА ДАНИХ, МЕСЕНДЖЕР, СЕРВЕР, ПОВІДОМЛЕННЯ, КЛІЄНТ, ЧАТ, PYTHON, SQLALCHEMY, ORM, POSTGRESQL, SQL, PGADMIN, TELEGRAM BOT API, ALEMBIC.

					«Житомирська політехніка».22.121.7.000 – ІПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата	Курсова робота «Розробка сервісу опитування з використанням Bot API від месенджера "Telegram"»		
Розроб.		Каліновський Д. Ю.					
Керівник		Чижмотря О.В.					
Реценз.							
Н. Контр.							
Затверд.		Чижмотря О.В.					
					Літ.	Арк.	Аркуші
						4	62
					ФІКТ, гр. ІПЗк-21-1		

ЗМІСТ

ВСТУП	6
РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТОКІВ ТА ОСОБЛИВОСТЕЙ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ	8
1.1 Аналіз інформаційних потреб та визначення предметної області дослідження	8
1.2 Обґрунтування вибору засобів реалізації	12
Висновки до першого розділу	13
РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ СИСТЕМИ ОПИТУВАНЬ	14
2.1 Аналіз інформаційних процесів системи опитувань	14
2.2 Проектування структури бази даних системи опитувань	16
2.3 Обробка даних системи	21
Висновки до другого розділу	23
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПІДСИСТЕМИ ОБРОБКИ ДАНИХ СИСТЕМИ ОПИТУВАНЬ	24
3.1 Проектування інтерфейсу обробки даних	24
3.2 Реалізація операцій обробки даних чат-ботом	28
3.3 Реалізація організації звітності системи	34
Висновки до 3 розділу:	37
РОЗДІЛ 4. АДМІНІСТРУВАННЯ БАЗ ДАНИХ	38
4.1 Розробка заходів захисту інформації в БД	38
4.2 Налаштування параметрів роботи PostgreSQL	40
Висновки до 4 розділу:	41
ВИСНОВКИ	42
ЛІТЕРАТУРА	43
ДОДАТКИ	44

ВСТУП

Актуальність теми. У наш час, коли думка мільйонів людей деформується під впливом стрімких і болючих змін і у той же час молодь не може навчатися у закладах освіти через постійну загрозу їх життю у місцях скупчення людей, особливо актуальним є додаток, що не потребує швидкого інтернету та великих затрат часу на налаштування і дозволяє проводити як соціальні опитування, так і тестування з вирахуванням оцінки.

Метою дослідження даної курсової роботи є розробка бази даних за допомогою ORM для використання у чат-боті для проведення опитувань.

Завданням дослідження було реалізувати базу даних для Telegram API чат-бота, яка буде містити в собі опитування та спроби їх проходження користувачами, а також оцінювати ці спроби, якщо це потрібно.

Об'єктом дослідження є розробка бази даних для чат-бота реляційної СУБД – PostgreSQL за допомогою мови запитів SQL, а також SQLAlchemy ORM для пришвидшеного формування запитів та спрощеної роботи з нею.

Предметом дослідження є вивчення можливостей взаємодії PostgreSQL та SQLAlchemy для коректної роботи з даними у рамках роботи Telegram-боту, що проводить опитування.

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижемотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – База даних.

СУБД – система управління базами даних.

КР – курсова робота.

ПЗ – Програмне забезпечення.

ORM – Object-Relational Mapping – Об'єктно-реляційна проєкція.

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижмоторя О.В.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТОКІВ ТА ОСОБЛИВОСТЕЙ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1 Аналіз інформаційних потреб та визначення предметної області дослідження

Задача полягає у створенні БД для сервісу опитувань із використанням Telegram Bot API. Оскільки єдина мова, на якій реалізовано бібліотеки для взаємодії із Telegram API, що є актуальними та постійно оновлюються – Python, то її і було обрано для виконання роботи. Для взаємодії з API обрано бібліотеку python-telegram-bot, що має найбільший об'єм коду серед конкурентів та оновлюється одним із перших. Для теми роботи актуальною є реляційні СУБД, тому була обрана одна з кращих на ринку і краща серед конкурентів із відкритим кодом – PostgreSQL. Оскільки система працює із SQL запитамі, що експоненційно збільшуються відповідно до ускладнення створюваного рішення, прийнято рішення використовувати бібліотеку SQLAlchemy та її ORM, що значно спрощують виконання поставлених задач та скорочують час, необхідний на їх виконання. Для створення та управління міграціями обрано Alembic, що чудово гармоніє із вже обраною бібліотекою для взаємодії із СУБД.

Відповідно до завдання сформовано наступні функції, що підлягають реалізації:

1. Створення нових опитувань різних типів (соціальне опитування, тестування).
2. Перегляд власних опитувань.
3. Редагування та видалення власних опитувань.
4. Пошук існуючих опитувань серед публічних.
5. Проходження опитувань, що є активованими.
6. Перегляд спроби проходження опитування та розширеного звіту по ній.
7. Резервне копіювання та відновлення адміністратором.

На сьогоднішній день сервісів для проведення опитувань є не дуже багато, однак існуючі аналоги є дуже якісними та мають широкий набір функцій. Перед тим як розглянути їх варто зауважити, що всіх них є великий мінус: вони мають закритий

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижмостря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

Розглянемо три найближчих найпопулярніших та найближчих за тематикою додатки:

Доступний за посиланням: <https://t.me/QuizBot>

Quiz Bot

Тестове опитування 1 16:17 ✓

Будь ласка, надішліть опис для вашого тесту. Цей крок можна пропустити: /skip. 16:17

/skip 16:17 ✓

Надішліть мені перше запитання вашого тесту у вигляді опитування з варіантами відповідей. Ви також можете надіслати повідомлення з текстом чи медіа, що показуватиметься перед запитанням.

Важливо: цей бот не може створювати анонімних опитувань. При проходженні тесту в групах, інші учасники завжди бачитимуть, хто голосував за який варіант. 16:17

До якої категорії за віком ви відноситься? 16:21 ✓

Denys
До якої категорії за віком ви відноситься?
Чудово. Тепер натисніть на кнопку, щоб надіслати мені питання, пов'язане із цим повідомленням.

Якщо ви надіслали повідомлення помилково, надішліть /undo для скасування. 16:21

До якої категорії за віком ви відноситься?
Анонімна вікторина

- ☐ 0-14
- ☐ 15-21
- ☐ 22-50
- ☐ 50-120

Немає відповідей 16:23 ✓

Добре, у тесті «Тестове опитування 1» тепер 1 запитання. Якщо Ви зробили помилку в запитанні, надішліть /undo.

Тепер надішліть наступне запитання. Ви також можете надіслати повідомлення з текстом чи медіа, яке показуватиметься перед запитанням.

Коли тест буде готовий, надішліть /done, щоб опублікувати його. 16:23

Меню

Написати повідомлення...

Переваги:

- 1) Базується на Telegram API, яким користуються мільйони людей, тобто не потребує ще однієї реєстрації від користувача.

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижомотря О.В.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

- 2) Використовує усі найсучасніші можливості програмного інтерфейсу Telegram, через що має дуже просту та приємну для користувача взаємодію.
- 3) Дозволяє проводити опитування у різних чатах та проглядати спрощену статистику (без графічних представлень)

Недоліки:

- 1) Дозволяє проводити тільки тестування без оцінювання, тобто опитування без вірних відповідей провести неможливо як і отримати оцінку за проходження.
- 2) Отримані результати тестування можна проглянути лише у текстовому та примітивному графічному виглядах.

2. Платформа Typeform

Доступна за посиланням <https://www.typeform.com/>

Зовнішній вигляд при створенні опитування:

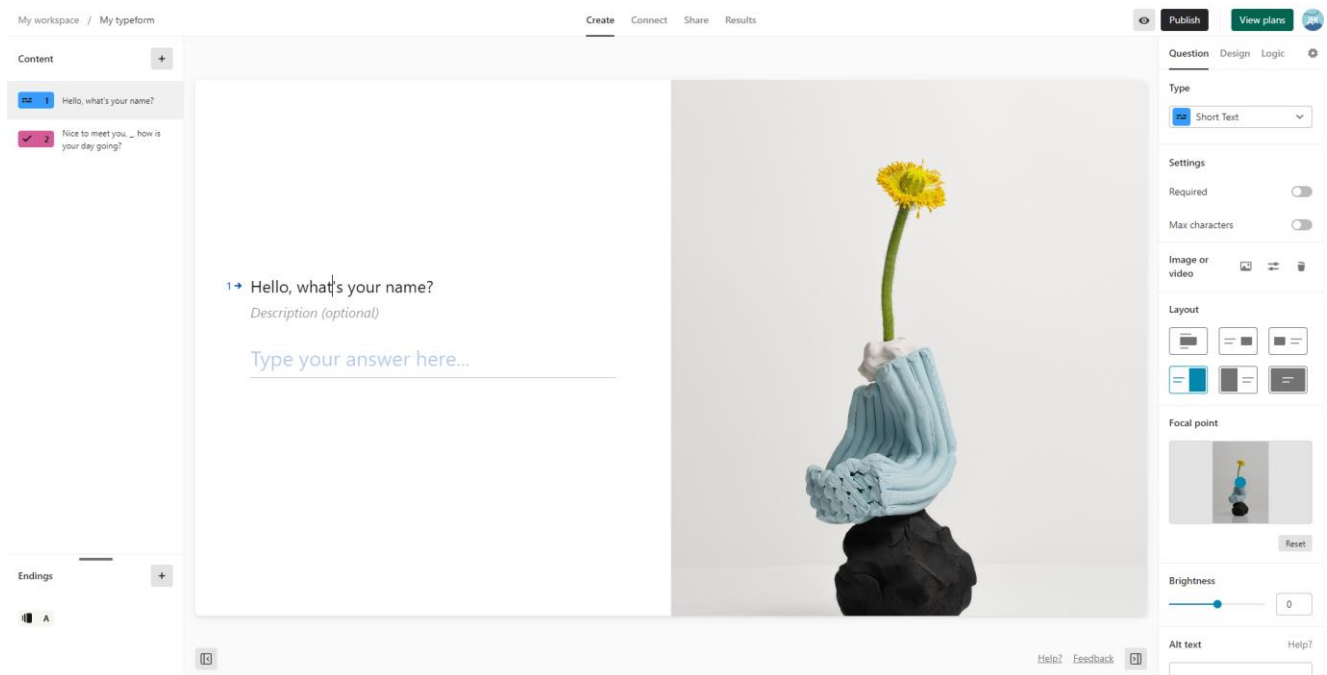


Рис. 1.2. Створення опитування у Typeform.

Переваги:

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

- 1) Має дуже широкий вибір типів запитань.
- 2) Має найбільші серед конкурентів можливості стилізації опитувань.
- 3) Є найбільш ергономічним та привітливим до користувача серед конкурентів.

Недоліки:

- 1) Створення опитування потребує багато часу.
- 2) Хоча й має великі можливості стилізації та налаштування, в конкретних функціях є дуже вузьким та незручним для автора.
- 3) Безкоштовно можна лише проглядати відповіді користувачів та примітивну інфографіку по ним.
- 4) Можна лише проводити опитування, тобто тестування із оцінюванням не реалізовано.

3. Веб-сайт Quiz Maker.

Доступний за посиланням: <https://www.quiz-maker.com/>

Зовнішній вигляд:

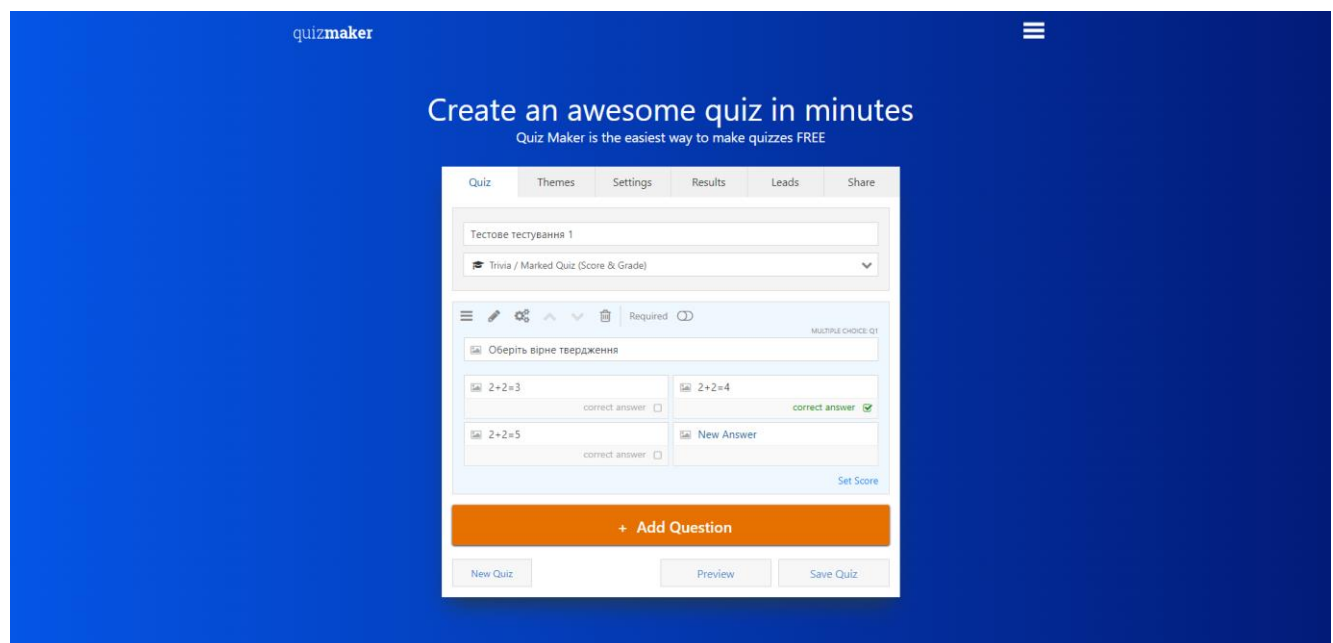


Рис. 1.3. Сайт Quiz Maker.

Переваги:

- 1) Простота використання як для автора, так і для користувача.
- 2) Широкий вибір шаблонів для різних типів опитувань. Єдиний аналог, що підтримує як різні типи опитувань, так і широкий набір видів тестувань.
- 3) Гнучке налаштування оцінювання та проходження тесту.
- 4) Найбільший набір статистичної інформації навіть у безкоштовному тарифі.

Недоліки:

- 1) Неергономічний інтерфейс, що неодмінно призведе до із використанням у користувачів.
- 2) Технічний вигляд, неприємний для звичайного користувача.

1.2 Обґрунтування вибору засобів реалізації

Ключовими у виборі СУБД були наступні два фактори:

- 1) СУБД повинна мати відкритий джерельний код;
- 2) СУБД повинна бути реляційною через те, що у проекті буде використано об'єктно-орієнтований підхід та об'єкти будуть щільно пов'язані між собою.

Таблиця 1.1

ХАРАКТЕРИСТИКА	POSTGRESQL	MARIADB	SQLITE
МОДЕЛЬ ДАНИХ	Реляційна	Реляційна	Реляційна
АДМІНІСТРАТИВНЕ КЕРУВАННЯ	Відмінно	Добре	Задовільно
ГРАФІЧНІ ІНСТРУМЕНТИ	Відмінно	Добре	Задовільно

ПРОСТОТА ОБСЛУГОВУВАННЯ	Відмінно	Відмінно	Відмінно
РОБОТА З ДЕКІЛЬКОМА ЦП	Відмінно	Відмінно	Задовільно
ПІДКЛЮЧЕННЯ ДО WEB	Добре	Відмінно	Добре
ПОБУДОВА БД	Добре	Добре	Задовільно
МОВА SQL	Відмінно	Добре	Задовільно
ІНТЕГРАЦІЯ З ІНШИМИ СУБД	Добре	Добре	Відмінно
ОДНОЧАСНИЙ ДОСТУП ДЕКІЛЬКОХ КОРИСТУВАЧІВ	Відмінно	Відмінно	Задовільно
МАСШТАБ ЗАСТОСУВАННЯ	Підприємства малих, середніх та великих розмірів	Підприємства малих, середніх розмірів	Підприємства малих розмірів

Було вирішено обрати реляційну СУБД PostgreSQL. Причини наведені нижче:

- Наявність докладної документації із прикладами;
- Ліцензія, що дозволяє необмежене використання;
- Активна спільнота, що розробляє та підтримує сотні додатків, що покращують життя як розробника так і адміністратора БД;
- Активна розробка, яку може контролювати та долучатися хто завгодно.

Висновки до першого розділу

Було проаналізовано задачу та функціонал, що потрібно буде реалізувати у межах даної курсової роботи. Проведено аналіз існуючих додатків, виділено їх переваги та недоліки, що будуть взяті до уваги під час розробки. Проаналізовано 3 популярні реляційні СУБД із відкритим джерельним кодом і обрано ту, що виявилася найкращою та що має найбільший об'єм функціональних можливостей.

- b. Обирає тип опитування (опитування чи тестування);
- c. Вводить категорії опитування;
- d. Вводить запитання або за бажанням переходить до останнього пункту якщо створено бодай одне запитання;
- e. Обирає кількість відповідей (одна або декілька);
- f. Вводить вірні відповіді (якщо створюється статистичне опитування то користувач вводить відповіді без прив'язки до правильності);
- g. Вводить не вірні відповіді (цей етап пропускається у випадку із статистичним опитуванням);
- h. Переходить до пункту d або до наступного;
- i. Користувач отримує інформацію про своє опитування (назва, код і тому подібне).

2. /my_quizzes – перегляд користувачем власних опитувань:

- a. Вибір опитування;
- b. Зміна будь-яких даних опитування, що не є незмінними (тип опитування), видалення опитування або перехід до пункту а;
- c. Перехід до списку запитань;
- d. Зміна будь-яких даних запитань, що не є незмінними (тип запитання), видалення запитання або перехід до пункту с;
- e. Перехід до списку відповідей;
- f. Зміна будь-яких даних запитання, видалення відповіді або перехід до пункту е.

3. /pass – проходження опитування за його кодом:

- a. Користувач обирає, чи хоче проходити це опитування;
- b. Користувач обирає відповідь або відповіді (залежно від типу запитання) або переходить до іншого запитання. Цей пункт повторюється допоки користувач не відповість на останнє запитання, у цьому випадку користувач переходить до пункту с;

- с. Користувач отримує інформацію про свою спробу проходження опитування (у тому числі унікальний код, за яким потім можна проглянути цю спробу).
- 4. /search – пошук опитування за частиною його назви (якщо опитування публічне) або його повним кодом:
 - а. Користувач може переходити по сторінкам, якщо опитувань, що підходять критеріям пошуку багато або обрати опитування для проходження. В останньому випадку користувач переходить до пункту 3.а.
- 5. /show – перегляд спроби за її унікальним кодом:
 - а. Користувач отримує повні відомості про спробу, однак без відповідей на конкретні запитання. Є можливість сформувати розгорнутий звіт із відповідями на запитання.
- 6. /backup – створити резервну копію бази даних, що використовується:
 - а. Якщо користувач є адміністратором, то створюється резервна копія та користувачу надсилаються короткі відомості про процес копіювання та саму копію.
- 7. /base_backup – створення базової резервної копії всього сервера:
 - а. Якщо користувач є адміністратором, то створюється базова резервна копія сервера та користувачу надсилаються короткі відомості про процес копіювання та саму копію.
- 8. /restore – відновити базу даних з резервної копії, що надав користувач:
 - а. Якщо користувач є адміністратором, то відбувається процес відновлення бази даних із файлу та користувачу виводяться короткі відомості про процес.

2.2 Проектування структури бази даних системи опитувань

Проектування бази даних відбувалося у вигляді моделей SQLAlchemy ORM, налагодження схеми даних проводилося в утиліті PgAdmin.

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

В рамках виконання роботи створено наступні таблиці:

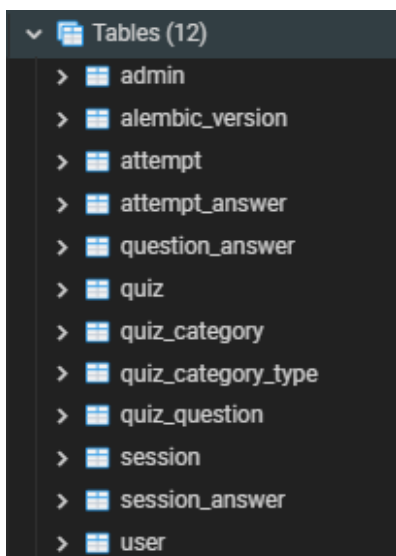


Рис. 2.2. Створені таблиці.

Кожна таблиця є результатом обробки SQLAlchemy ORM-моделі, прописаної у коді додатку. Такий підхід дозволяє значно спростити взаємодію із БД та зменшити об'єми і витрати часу на налаштування цієї взаємодії.

Таблиця, що є результатом обробки моделі User:

Columns									
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default		
	id	bigint			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
	data	json			<input type="checkbox"/>	<input type="checkbox"/>			

Рис. 2.3. Таблиця user.

Таблиця, що є результатом обробки моделі Quiz:

Columns								+
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default	
		id	integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('qui
		name	character varying	50		<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		author_id	integer			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		is_public	boolean			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		is_statistical	boolean			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		is_available	boolean			<input checked="" type="checkbox"/>	<input type="checkbox"/>	true
		token	character varying	10		<input checked="" type="checkbox"/>	<input type="checkbox"/>	'basedbase

Рис. 2.4. Таблиця quiz.

Таблиця, що є результатом обробки моделі QuizQuestion:

Columns								+
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default	
		id	bigint			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('qui
		quiz_id	integer			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		question	character varying	256		<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		is_multi	boolean			<input checked="" type="checkbox"/>	<input type="checkbox"/>	

Рис. 2.5. Таблиця quiz_question.

Таблиця, що є результатом обробки моделі QuizCategoryType:

Columns								+
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default	
		id	smallint			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('qui
		name	character varying	50		<input checked="" type="checkbox"/>	<input type="checkbox"/>	

Рис. 2.6. Таблиця quiz_category_type.

Таблиця, що є результатом обробки моделі QuizCategory:

Columns								+
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default	
		id	bigint		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('qu	
		quiz_id	integer		<input checked="" type="checkbox"/>	<input type="checkbox"/>		
		category_id	smallint		<input checked="" type="checkbox"/>	<input type="checkbox"/>		

Рис. 2.7. Таблиця quiz_category.

Таблиця, що є результатом обробки моделі QuestionAnswer:

Columns								+
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default	
		id	bigint		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('qu	
		question_id	integer		<input checked="" type="checkbox"/>	<input type="checkbox"/>		
		answer	character varying	256	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
		is_right	boolean		<input checked="" type="checkbox"/>	<input type="checkbox"/>		

Рис. 2.8. Таблиця question_answer.

Таблиця, що є результатом обробки моделі Admin:

Columns								+
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default	
		user_id	bigint		<input checked="" type="checkbox"/>	<input type="checkbox"/>		

Рис. 2.9. Таблиця admin.

Таблиця, що є результатом обробки моделі Session:

Columns								+
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default	
		id	bigint		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('se:	
		user_id	bigint		<input checked="" type="checkbox"/>	<input type="checkbox"/>		
		quiz_id	integer		<input checked="" type="checkbox"/>	<input type="checkbox"/>		
		started_on	timestamp without time...		<input checked="" type="checkbox"/>	<input type="checkbox"/>		
		question_number	integer		<input checked="" type="checkbox"/>	<input type="checkbox"/>		

Рис. 2.10. Таблиця session.

Таблиця, що є результатом обробки моделі SessionAnswer:

Columns							
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
	id	bigint			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('se
	session_id	bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	question_id	integer			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	answer_ids	bigint[]			<input checked="" type="checkbox"/>	<input type="checkbox"/>	

Рис. 2.11. Таблиця session_answer.

Таблиця, що є результатом обробки моделі Attempt:

Columns							
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
	id	bigint			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('att
	user_id	bigint			<input type="checkbox"/>	<input type="checkbox"/>	
	quiz_id	integer			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	started_on	timestamp without time..			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	finished_on	timestamp without time..			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	mark	numeric			<input type="checkbox"/>	<input type="checkbox"/>	
	uuid	uuid			<input type="checkbox"/>	<input type="checkbox"/>	
	report_path	character varying			<input type="checkbox"/>	<input type="checkbox"/>	

Рис. 2.12. Таблиця attempt.

Таблиця, що є результатом обробки моделі AttemptAnswer:

Columns							
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
	id	bigint			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('att
	attempt_id	bigint			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	question_id	integer			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	answer_ids	bigint[]			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
	mark	numeric			<input type="checkbox"/>	<input type="checkbox"/>	

Рис. 2.13. Таблиця attempt_answer.

Технічна таблиця, що зберігає у собі ідентифікатор поточної версії міграції. Цю таблицю створює Alembic, і він же її адмініструє. Схема цієї таблиці має наступний вигляд:

Columns							
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
	version_num	character varying	32		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Рис. 2.14. Таблиця alembic_version.

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижомотря О.В.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

вірних та частково вірних відповідей, даних користувачем та кількість запитань, що користувач пропустив. Лістинг отримання та обробки цих даних:

```
with db_session.begin() as s:
    attempt = s.get(Attempt, attempt_id)
    right_count = s.query(AttemptAnswer).filter(and_(
        AttemptAnswer.attempt_id == attempt_id,
        AttemptAnswer.mark == Decimal('1'))).count()
    partially_right_count = s.query(AttemptAnswer).filter(and_(
        AttemptAnswer.attempt_id == attempt_id,
        and_(AttemptAnswer.mark > Decimal('0'), AttemptAnswer.mark <
Decimal('1')))).count()
    wrong_count = s.query(AttemptAnswer).filter(and_(
        AttemptAnswer.attempt_id == attempt_id,
        AttemptAnswer.mark == Decimal('0'))).count()
    unanswered_count =
s.query(QuizQuestion).filter_by(quiz_id=attempt.quiz_id).count() - \
    (right_count + wrong_count + partially_right_count)
```

Ця взаємодія із даними є достатньо простою, оскільки включає в себе лише запити на кількість записів-відповідей у рамках наданої спроби на запитання, що мають оцінки 0, 1 та щось посередині. Для отримання четвертої категорії (кількості пропущених запитань) отримується загальна кількість запитань та від неї віднімаються вже вираховані значення. У результаті маємо чотири значення, готові для переведення у графічне представлення.

Другою такою операцією, вартою згадки, є вирахування кількості проходжень опитування за різні періоди часу та за різні часові проміжки. Лістинг відповідного цієї операції коду:

```
with db_session.begin() as s:
    stats = s.execute(text(
        f""""SELECT DATE_TRUNC(:part, att.started_on), COUNT(*) as "Number of attempts"
        FROM attempt att
        WHERE :quiz_id = att.quiz_id """" +
        (f"AND DATE_PART('day', DATE_TRUNC(:period, NOW())) - DATE_TRUNC(:period,
att.started_on)) <= {time_parts_day_eq[period]}" if period != 'all' else "") +
        """" GROUP BY DATE_TRUNC(:part, att.started_on)
        ORDER BY DATE_TRUNC(:part, att.started_on);
        """"), {'quiz_id': quiz_id, 'part': part, 'period': period}).all()
```

Цього разу для вирахування одним запитом використано SQL-запит, що формується залежно від наданих користувачем даних. Наприклад, якщо користувач захоче звіт за весь час, то код просто не внесе в запит частину, що обмежує період часу, за який формується звіт.

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

Вартим згадки також є запит, що формує статистику відповідей на запитання для усіх запитань опитування. Лістинг коду, що утворює цей запит та оброблює дані, що приходять на нього від СУБД:

```
with db_session.begin() as s:
    titles = [x[0] for x in
s.query(QuizQuestion.question).filter_by(quiz_id=quiz_id).all()]
    question_ids = s.query(QuizQuestion.id).filter_by(quiz_id=quiz_id).all()

    for que_id_index in range(len(question_ids)):
        que_id = question_ids[que_id_index][0]
        title = titles[que_id_index]
        labels = [x[0] for x in
s.query(QuestionAnswer.answer).filter_by(question_id=que_id).all()]
        answer_ids = [x[0] for x in
s.query(QuestionAnswer.id).filter_by(question_id=que_id).all()]
        stmt = text(
            "SELECT question_answer.answer, COUNT(question_answer.id) \n"
            "FROM question_answer \n"
            "INNER JOIN attempt_answer aa ON question_answer.id=ANY(aa.answer_ids) \n"
            "INNER JOIN attempt att ON att.id = aa.attempt_id \n" +
            (f"AND DATE_PART('day', DATE_TRUNC(:period, NOW())) - DATE_TRUNC(:period,
att.started_on)) <= {time_parts_day_eq[period]} \n" if period != 'all' else "") +
            "WHERE question_answer.id = ANY(:ans_ids) \n"
            "GROUP BY question_answer.id;")

        counts = s.execute(stmt, {'ans_ids': answer_ids, 'period': period}).all()
```

Цей запит формує дані відношення кількості вибору відповідей на кожне запитання, обмежені за часом (або не обмежені, залежно від запиту користувача), одразу готові до відображення на графіках.

Висновки до другого розділу

Складено алгоритм взаємодії користувача із чат-ботом у подробицях. Проведено аналіз та детально продумано можливості як звичайного користувача, так і привілейованого.

Відповідно до особливостей інформаційних процесів у додатку, створено моделі для всіх необхідних у стабільній роботі сутностей. На основі створених моделей згенеровано таблиці та діаграму бази даних, що буде використана у цій роботі

Створено операції, що аналізують дані та приводять результати цього аналізу у доступний користувачу вигляд. У результаті роботи цього коду користувач отримає графічно подану інформацію щодо своїх опитувань або спроби проходження тестування.

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПІДСИСТЕМИ ОБРОБКИ ДАНИХ СИСТЕМИ ОПИТУВАНЬ

3.1 Проектування інтерфейсу обробки даних

У обраній темі роботи проектування інтерфейсу у традиційному його визначенні не є можливим, оскільки у якості інтерфейсу використовується інтерфейс месенджера Telegram. У зв'язку із цим простір проектування звужується до трьох елементів: форматування тексту, прикріплення рисунків до повідомлень і клавіатура повідомлення. Всі три способи використано у роботі на повну.

Перший елемент, форматування тексту, використано у всіх елементах взаємодії із користувачем. Далі наведено ті, з якими користувач найчастіше зустрічається:

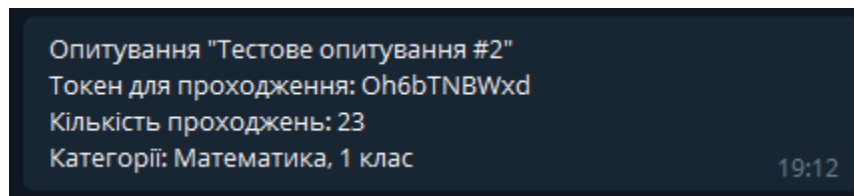


Рис. 3.1. Інформація про обране опитування.

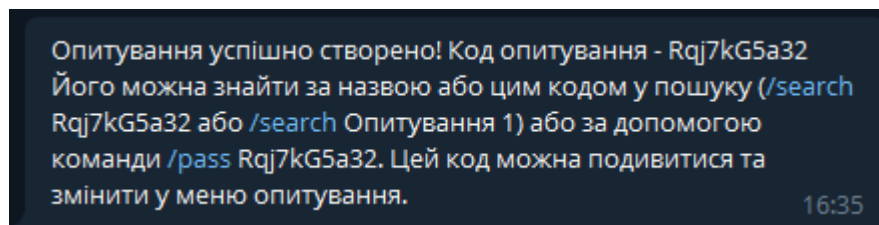


Рис. 3.2. Інформація про новостворене опитування.

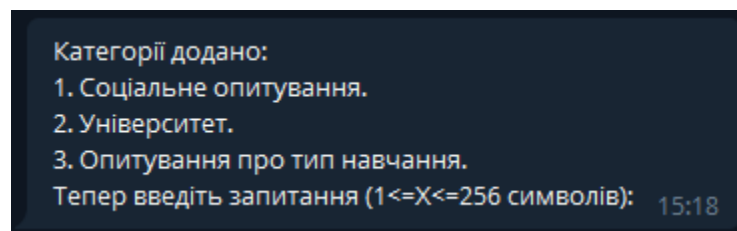


Рис. 3.3. Інформація про додані користувачем до опитування категорії.

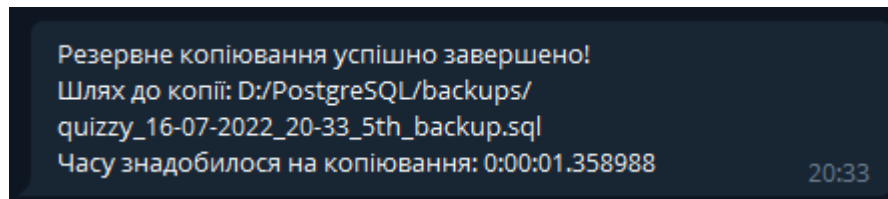


Рис. 3.4. Інформація про створену резервну копію.

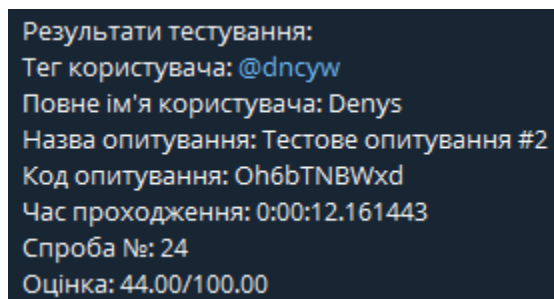


Рис. 3.5. Інформація про результати тестування.

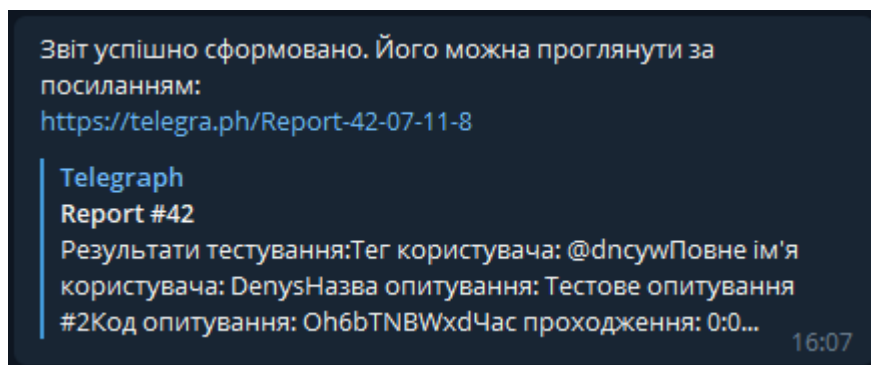


Рис. 3.6. Відповідь на запит користувача про створення розгорнутого.

Клавіатура повідомлення також широко використовується, оскільки, по суті, є єдиним способом взаємодії із користувачем, не засмічуючи чат сотнями повідомлень, що між собою майже не відрізняються. Приклади використання цього елементу дизайну:

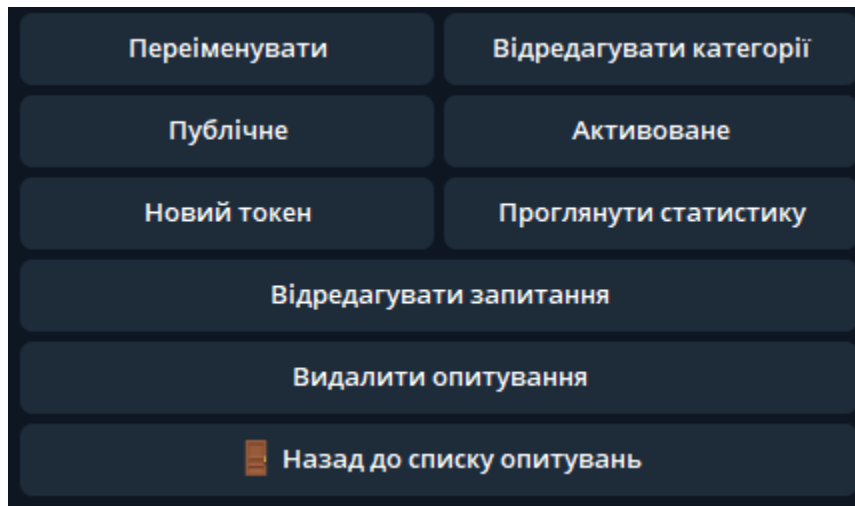


Рис. 3.7. Клавіатура редагування опитування.

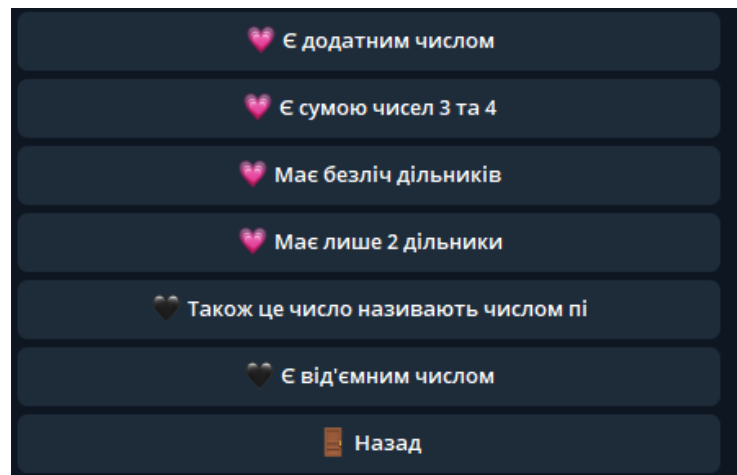


Рис. 3.8. Клавіатура вибору відповіді для редагування.

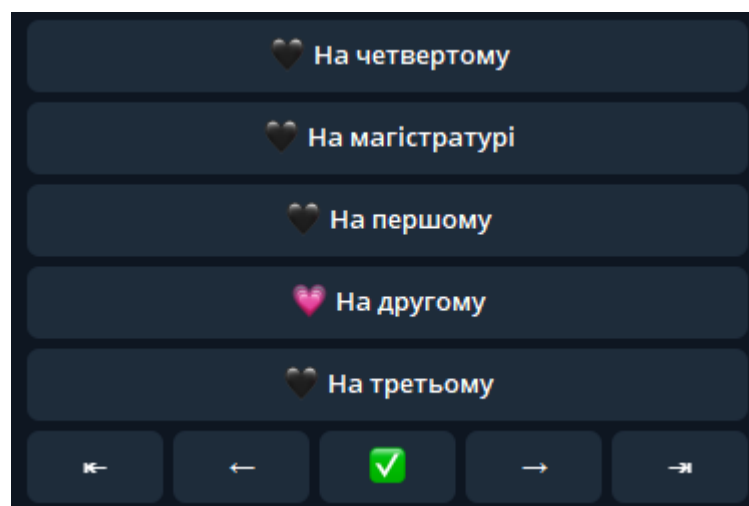


Рис. 3.9. Клавіатура проходження опитування.

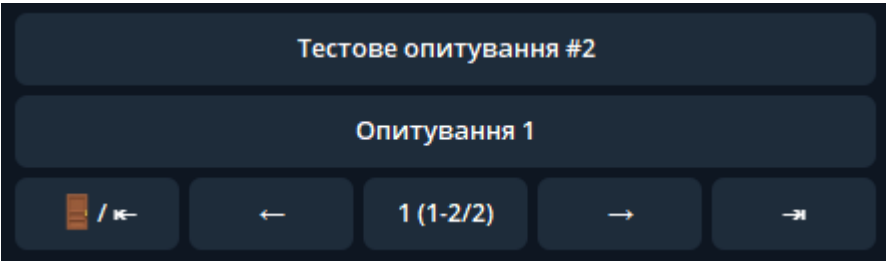


Рис. 3.10. Клавіатура пошуку опитування.

Надсилання медіа є найменш гнучким з наявних варіантів, однак водночас і найбільш наочним, через що й використовується у цій роботі лише для графічного відображення статистики. Приклади такого використання:



Рис. 3.11. Відображення статистики проходження опитування.

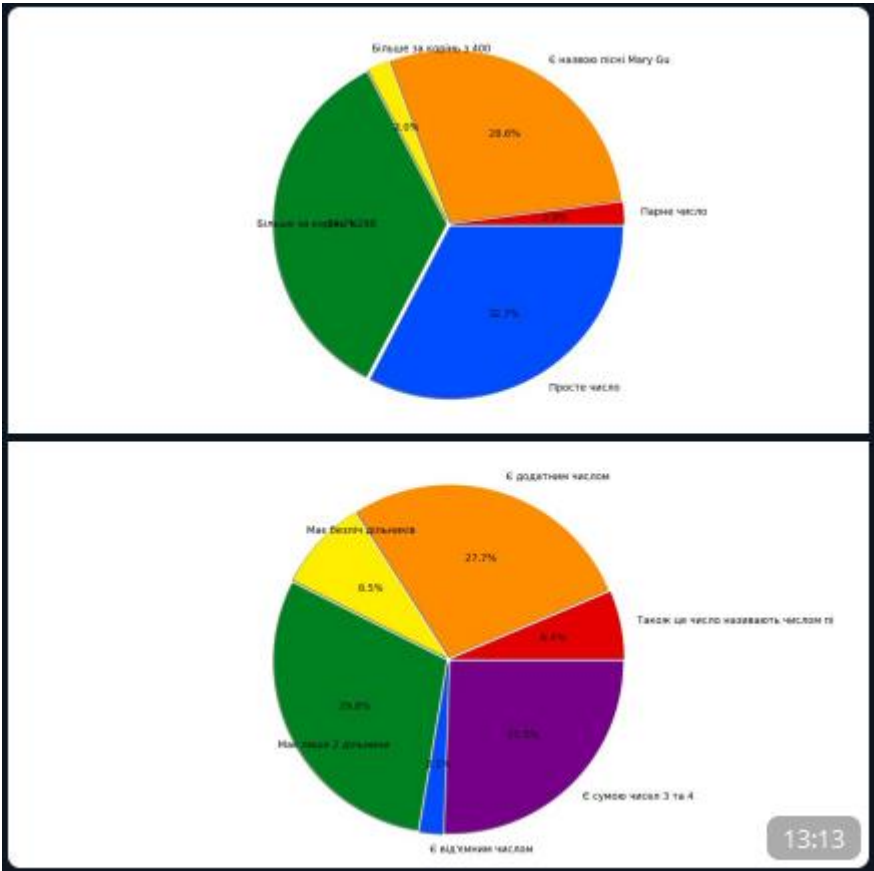


Рис. 3.12. Відображення статистики обрання певних відповідей.

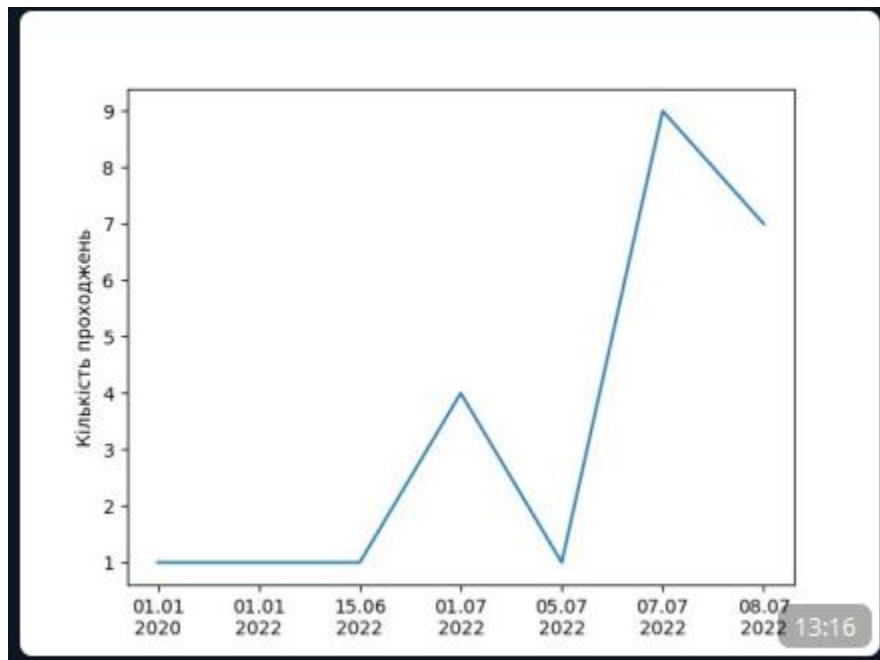


Рис. 3.13. Відображення кількості проходжень опитування у різні дні.

3.2 Реалізація операцій обробки даних чат-ботом

У зв'язку із необхідністю встановлення абстракції роботи із базою даних, взаємодія із БД та обробка даних здійснюється зв'язкою двох бібліотек:

1. `psycopg3` – є драйвером СУБД, тобто виконує із нею всі операції, отримує із неї дані та оброблює помилки та повідомлення, отримані від неї.
2. `SQLAlchemy` – є менеджером SQL, тобто генерує запити до БД та видозмінює відповідь від неї таким чином, щоб із ними було зручно працювати у Python.

Оскільки друга бібліотека займається обробкою даних, що приходять із БД, то саме її можливості буде використано для цього. А саме, буде використано можливість створення «моделей», що являють собою таблиці у базі даних, доповнені функціями мови, на якій вони розроблені. Розроблені моделі мають наступний вигляд:

1. Лістинг моделі User:

```
from sqlalchemy import Column, BigInteger, JSON
from sqlalchemy.orm import relationship, backref
from sqlalchemy.orm.attributes import flag_modified

from db.engine import BaseModel
```

```

class User(BaseModel):
    __tablename__ = 'user'

    def __init__(self, user_id):
        self.id = user_id

    def flag_data(self):
        flag_modified(self, 'data')

    def set_data(self, key, value):
        self.data[key] = value
        flag_modified(self, 'data')

    def remove_data(self, key):
        del self.data[key]
        flag_modified(self, 'data')

    @property
    def is_admin(self):
        print(self.admin)
        return self.admin is not None

    id = Column(BigInteger, autoincrement=False, primary_key=True)
    data = Column(JSON, default={})
    current_session = relationship('Session', backref=backref('user', lazy='select'),
uselist=False,
                                cascade='all, delete, delete-orphan')
    quizzes = relationship('Quiz', backref=backref('author', lazy='select'),
                                cascade='all, delete, delete-orphan')
    attempts = relationship('Attempt', backref=backref('user', lazy='select'),
                                cascade='all, delete, delete-orphan')

```

2. Лістинг моделі Quiz:

```

from sqlalchemy import Column, Integer, String, ForeignKey, Boolean, BigInteger
from sqlalchemy.orm import relationship, backref
from sqlalchemy.ext.hybrid import hybrid_property

from db.models.User import User
from db.engine import BaseModel, db_session

from utils import generate_token

class Quiz(BaseModel):
    __tablename__ = 'quiz'

    def __init__(self, name, author_id, is_public, is_statistical):
        self.name = name
        self.author_id = author_id
        self.is_public = is_public
        self.is_statistical = is_statistical

        self.regenerate_token()

    def __repr__(self):
        return f'<Quiz: {{"name": {self.name}, author_id={self.author_id}, is_public={self.is_public}, ' \
            f'is_statistical={self.is_statistical}, token={self.token}}}>'

    @hybrid_property

```

```

def categories(self):
    return [x.name for x in self.categories_ref]

def regenerate_token(self):
    token = generate_token()
    with db_session.begin() as s:
        while s.query(Quiz.id).filter(Quiz.token == token).one_or_none() is not
None:
            token = generate_token()
    self.token = token

id = Column(Integer, primary_key=True)
name = Column(String(50), nullable=False)
author_id = Column(BigInteger, ForeignKey(User.id, ondelete='CASCADE'),
nullable=False)
is_available = Column(Boolean, default=True, nullable=False)
is_public = Column(Boolean, default=True, nullable=False)
is_statistical = Column(Boolean, default=False, nullable=False)
token = Column(String(10), unique=True, nullable=False)

questions = relationship('QuizQuestion', backref=backref('quiz', lazy='select'),
                        cascade='all, delete, delete-orphan')
categories_ref = relationship('QuizCategoryType',
                             secondary='quiz_category',
                             primaryjoin='Quiz.id==QuizCategory.quiz_id',
secondaryjoin='QuizCategoryType.id==QuizCategory.category_id')

```

3. Лістинг моделі QuizCategoryType:

```

from sqlalchemy import Column, SmallInteger, String

from db.engine import BaseModel

class QuizCategoryType(BaseModel):
    __tablename__ = 'quiz_category_type'

    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return f'<QuizCategoryType id: {self.id}, name: {self.name}>'

    id = Column(SmallInteger, primary_key=True)
    name = Column(String(50), nullable=False, unique=True)

```

4. Лістинг моделі QuizCategory:

```

from sqlalchemy import Column, BigInteger, Integer, ForeignKey, SmallInteger,
UniqueConstraint

from db.models.Quiz import Quiz
from db.models.QuizCategoryType import QuizCategoryType
from db.engine import BaseModel

class QuizCategory(BaseModel):
    __tablename__ = 'quiz_category'

    def __init__(self, quiz_id, category_id):

```

```

        self.quiz_id = quiz_id
        self.category_id = category_id

    id = Column(BigInteger, primary_key=True)
    quiz_id = Column(Integer, ForeignKey(Quiz.id, ondelete='CASCADE'), nullable=False)
    category_id = Column(SmallInteger, ForeignKey(QuizCategoryType.id,
ondelete='CASCADE'), nullable=False)
    UniqueConstraint(quiz_id, category_id, name='unique_cat')

```

5. Лістинг моделі QuizQuestion:

```

from sqlalchemy import Column, BigInteger, Integer, ForeignKey, String, Boolean
from sqlalchemy.orm import relationship, backref

from db.models.Quiz import Quiz
from db.engine import BaseModel

class QuizQuestion(BaseModel):
    __tablename__ = 'quiz_question'

    def __init__(self, quiz_id, question, multi):
        self.quiz_id = quiz_id
        self.question = question
        self.is_multi = multi

    def __repr__(self):
        return f'<QuizQuestion: {{id: {self.id}, quiz_id: {self.quiz_id}, question: {self.question}}, ' \
            f'is_multi: {self.is_multi}}>'

    def __str__(self):
        return f'Запитання: {self.question}\n' + \
            f'Відповіді:\n' + \
            '\n'.join([str(x) for x in self.answers])

    id = Column(BigInteger, primary_key=True)
    quiz_id = Column(Integer, ForeignKey(Quiz.id, ondelete='CASCADE'), nullable=False)
    question = Column(String(256), nullable=False)
    is_multi = Column(Boolean, default=False, nullable=False)

    answers = relationship('QuestionAnswer', backref=backref('question',
lazy='select'),
                        cascade='all, delete, delete-orphan')

```

6. Лістинг моделі QuestionAnswer:

```

from sqlalchemy import Column, BigInteger, Integer, ForeignKey, String, Boolean

from db.models.QuizQuestion import QuizQuestion
from db.engine import BaseModel
from cfg import GOOD_SIGN, BAD_SIGN

class QuestionAnswer(BaseModel):
    __tablename__ = 'question_answer'

    def __init__(self, question_id, answer, is_right):
        self.question_id = question_id
        self.answer = answer
        self.is_right = is_right

```

```

def __str__(self):
    return f'{GOOD_SIGN if self.is_right else BAD_SIGN} {self.answer}'

id = Column(BigInteger, primary_key=True)
question_id = Column(Integer, ForeignKey(QuizQuestion.id, ondelete='CASCADE'),
nullable=False)
answer = Column(String(256), nullable=False)
is_right = Column(Boolean, default=False, nullable=False)

```

7. Лістинг моделі Admin:

```

from sqlalchemy import Column, BigInteger, ForeignKey

from db.engine import BaseModel
from db.models.User import User

class Admin(BaseModel):
    __tablename__ = 'admin'

    def __init__(self, user_id):
        self.user_id = user_id

    user_id = Column(BigInteger, ForeignKey(User.id), primary_key=True)

```

8. Лістинг моделі Session:

```

import datetime

from sqlalchemy import Column, BigInteger, ForeignKey, Integer, DateTime,
UniqueConstraint

from db.models.Quiz import Quiz
from db.models.User import User
from db.engine import db_session, BaseModel

class Session(BaseModel):
    __tablename__ = 'session'

    def __init__(self, user_id, quiz_id):
        self.user_id = user_id
        self.quiz_id = quiz_id

    id = Column(BigInteger, primary_key=True)
    user_id = Column(BigInteger, ForeignKey(User.id, ondelete='CASCADE'),
nullable=False, unique=True)
    quiz_id = Column(Integer, ForeignKey(Quiz.id, ondelete='CASCADE'), nullable=False)
    started_on = Column(DateTime, default=datetime.datetime.now, nullable=False)
    question_number = Column(Integer, default=-1, nullable=False)

```

9. Лістинг моделі SessionAnswer:

```

from sqlalchemy import Column, BigInteger, ForeignKey, Integer
from sqlalchemy.dialects.postgresql import ARRAY

from db.models.QuestionAnswer import QuestionAnswer
from db.models.QuizQuestion import QuizQuestion
from db.models.Session import Session
from db.engine import BaseModel

```



```

class SessionAnswer(BaseModel):
    __tablename__ = 'session_answer'

    def __init__(self, session_id, question_id, answer_ids: list):
        self.session_id = session_id
        self.question_id = question_id
        self.answer_ids = answer_ids.copy()

    id = Column(BigInteger, primary_key=True)
    session_id = Column(BigInteger, ForeignKey(Session.id, ondelete='CASCADE'),
nullable=False)
    question_id = Column(Integer, ForeignKey(QuizQuestion.id, ondelete='CASCADE'),
nullable=False)
    answer_ids = Column(ARRAY(BigInteger, as_tuple=ForeignKey(QuestionAnswer.id,
ondelete='CASCADE'))), nullable=False)

```

10. Лістинг моделі Attempt:

```

import datetime

from sqlalchemy import Column, BigInteger, ForeignKey, Integer, DateTime, Numeric,
String
from sqlalchemy.orm import relationship, backref
from sqlalchemy.dialects.postgresql import UUID

from db.models.Quiz import Quiz
from db.models.Session import Session
from db.models.User import User
from db.engine import BaseModel, db_session

import uuid

class Attempt(BaseModel):
    __tablename__ = 'attempt'

    def __init__(self, user_id, quiz_id, started_on):
        self.user_id = user_id
        self.quiz_id = quiz_id
        self.started_on = started_on

    @classmethod
    def from_session(cls, quiz_session: Session):
        return cls(quiz_session.user_id, quiz_session.quiz_id,
quiz_session.started_on)

    id = Column(BigInteger, primary_key=True)
    uuid = Column(UUID(as_uuid=True), unique=True, default=uuid.uuid4)
    user_id = Column(BigInteger, ForeignKey(User.id, ondelete='SET NULL'),
nullable=True)
    quiz_id = Column(Integer, ForeignKey(Quiz.id, ondelete='CASCADE'), nullable=False)
    started_on = Column(DateTime, nullable=False)
    finished_on = Column(DateTime, default=datetime.datetime.now, nullable=False)
    report_path = Column(String, nullable=True)
    answers = relationship('AttemptAnswer', backref=backref('attempt', lazy='select'),
cascade='all, delete, delete-orphan')
    mark = Column(Numeric, nullable=True, default=0, server_default='0')

```

11. Лістинг моделі AttemptAnswer:

```
from sqlalchemy import Column, BigInteger, ForeignKey, Integer, UniqueConstraint,
Numeric
from sqlalchemy.orm import relationship
from sqlalchemy.dialects.postgresql import ARRAY

from db.models.Attempt import Attempt
from db.models.QuestionAnswer import QuestionAnswer
from db.models.QuizQuestion import QuizQuestion
from db.models.SessionAnswer import SessionAnswer
from db.engine import BaseModel

class AttemptAnswer(BaseModel):
    __tablename__ = 'attempt_answer'

    def __init__(self, attempt_id, question_id, answer_ids: tuple):
        self.attempt_id = attempt_id
        self.question_id = question_id
        self.answer_ids = answer_ids

    def __repr__(self):
        return f'Id: {self.id}, attempt_id: {self.attempt_id}, question_id: {self.question_id}, ' \
            f'answer_ids: {self.answer_ids}'

    @classmethod
    def from_session_answer(cls, attempt_id: int, session_answer: SessionAnswer):
        return cls(attempt_id, session_answer.question_id,
            tuple(session_answer.answer_ids))

    id = Column(BigInteger, primary_key=True)
    attempt_id = Column(BigInteger, ForeignKey(Attempt.id, ondelete='CASCADE'),
        nullable=False)
    question_id = Column(Integer, ForeignKey(QuizQuestion.id, ondelete='CASCADE'),
        nullable=False)
    answer_ids = Column(ARRAY(BigInteger, as_tuple=ForeignKey(QuestionAnswer.id)),
        nullable=False)
    UniqueConstraint(attempt_id, question_id, name='unique_attempt_answer')
    mark = Column(Numeric, nullable=True, default=0, server_default='0')
```

3.3 Реалізація організації звітності системи

В проекті реалізовано декілька видів звітності, у тому числі ті, що вже частково були розглянуті в пунктах 2.3 та 3.1 та будуть розглянуті більш детально під час розгляду організації звітності системи.

Серед інших у системі звітність реалізовано у наступних місцях:

- 1) Відображення кількості проходжень, кількості людей, що прямо зараз проходять це опитування (відображається лише якщо такі є) та категорій цього опитування:

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

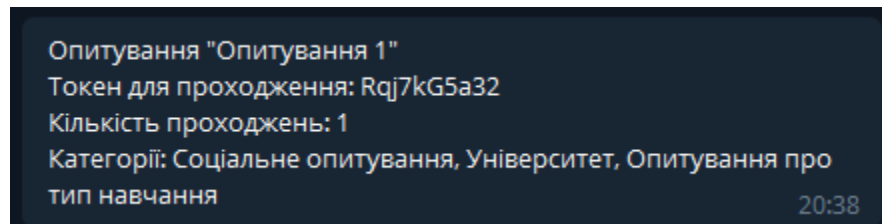


Рис. 3.14. Інформація про опитування, що належить користувачу.

- 2) Підрахунок оцінки за спробу при внесенні нової спроби або зміні запитань чи відповідей, що відбувається на стороні бази даних. Він є повністю непомітним для користувача, оскільки коли спробу внесено у БД, то і оцінка вже порахована.
- 3) Відображення результатів проходження тестування у вигляді горизонтального стовпчикowego графіку, часу проходження (що насправді не зберігається, а вираховується за потреби), номера спроби (що також вираховується безпосередньо перед виводом користувачу) та оцінки:

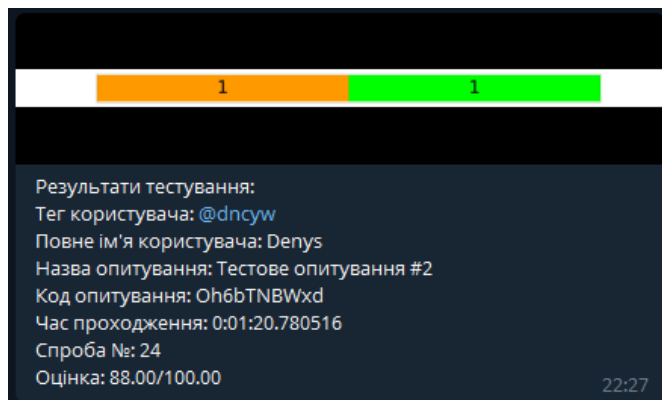


Рис. 3.15. Інформація про спробу проходження тестування.

- 4) Формування повного звіту по відповідям спроби проходження тестування:

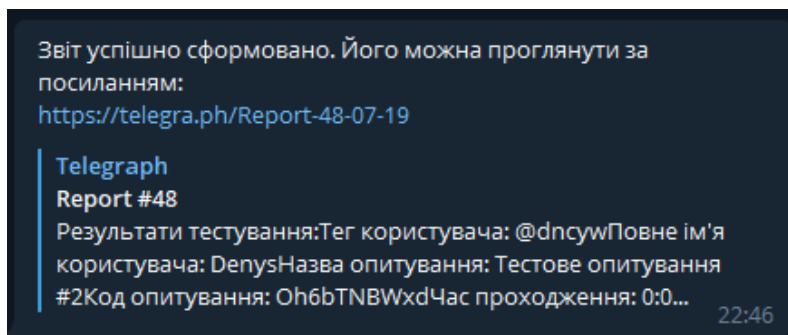


Рис. 3.16. Бот надсилає користувачу посилання на повний звіт.

Report #48

Quizzzy attempt report • July 19, 2022

Quizzzy

EDIT

Результати тестування:

Тег користувача: @dncyw

Повне ім'я користувача: Denys

Назва опитування: Тестове опитування #2

Код опитування: Oh6bTNBWxd

Час проходження: 0:01:20.780516

Спроба №: 24

Оцінка: 100.00/100.00

По запитанням:

[1.00/1.00] Оберіть вірні твердження про число 17

♥ Є назвою пісні Mary Gu

♥ Просте число

♥ Більше за корінь з 288

♥ Більше за корінь з 400

♥ Парне число

[1.00/1.00] Оберіть властивості числа 7

♥ Є додатним числом

♥ Є сумою чисел 3 та 4

♥ Має лише 2 дільники

♥ Є від'ємним числом

♥ Також це число називають числом пі

♥ Має безліч дільників

Рис. 3.17. Вигляд повного звіту, що знаходиться за посиланням на рис. 3.16.

5) Відображення статистики вибору відповідей кожного із запитань опитування за обраний період (весь час, рік, місяць) у вигляді діаграми-«пирога»:

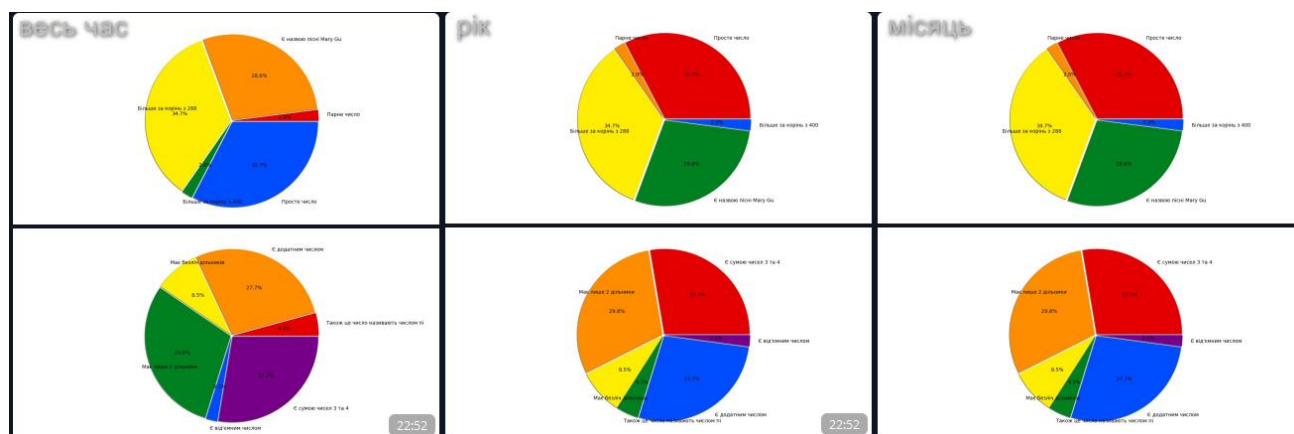


Рис. 3.18. Графіки за різні періоди часу.

6) Відображення статистики вибору відповідей кожного із запитань опитування за обраний період (весь час, рік, місяць) та розбитий на частини заданої довжини (день, місяць, рік) у вигляді графіка:

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижемська О. В.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

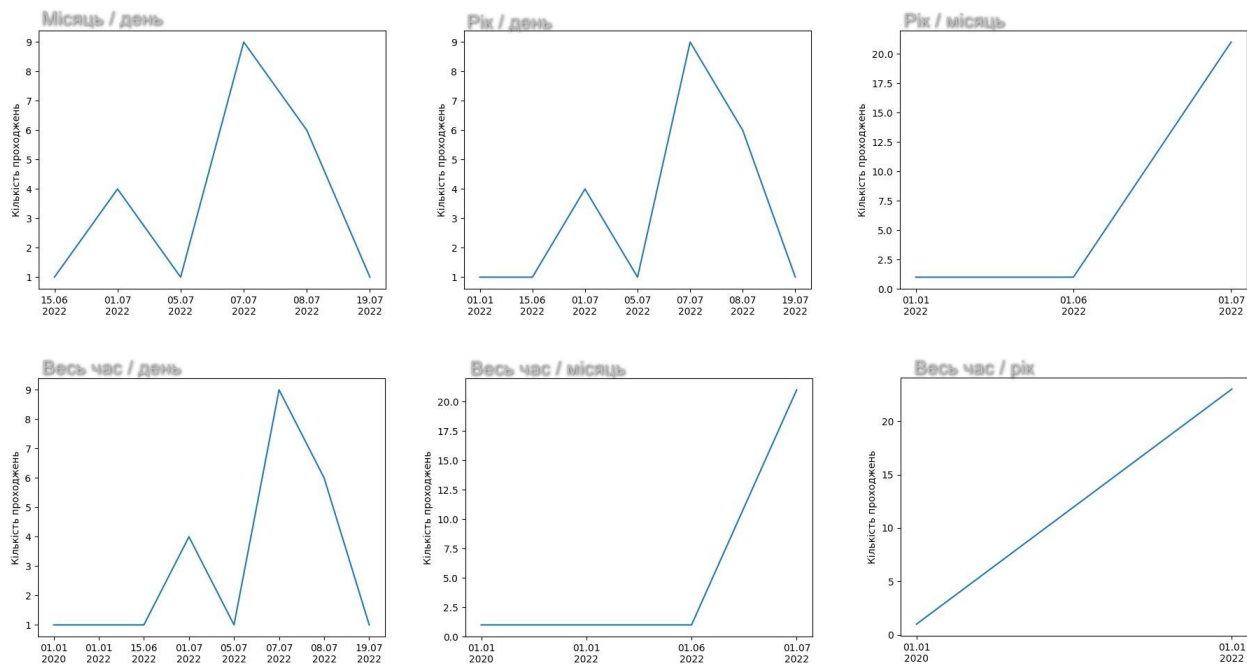


Рис. 3.19. Графіки за різні періоди часу та поділені на різні частини.

Висновки до 3 розділу:

Спроековано інтерфейс взаємодії із користувачем за допомогою наявних в Telegram API інструментів, себто текст повідомлення, клавіатура повідомлення та медіа-повідомлення.

Сформовано моделі для спрощення, пришвидшення та зменшення об'ємів коду, необхідного для взаємодії із базою даних.

Реалізовано звітність системи у виді агрегації записів (підрахунок спроб, обрахунок оцінок) та перетворення даних у графіки, що легко візуально сприймаються користувачем.

РОЗДІЛ 4. АДМІНІСТРУВАННЯ БАЗ ДАНИХ

4.1 Розробка заходів захисту інформації в БД

Захист інформації, що належить певним користувачам реалізовано за допомогою функціоналу Telegram API. Програмований інтерфейс реалізує реєстрацію та авторизацію користувача, разом із розділенням користувачів на індивідуальні чати, що дозволяє як поділяти користувачів, безпечно розміщуючи інформацію у індивідуальних чатах (таким чином надіслану користувачу інформацію може отримати інший користувач тільки у випадку, якщо перший перешле її другому), так і прив'язувати усі дані в базі даних до конкретних користувачів, як, наприклад, опитування, спроби проходження опитувань і так далі.

Таким чином, завдяки системі авторизації Telegram API, всі опитування користувача, як і інші дані, що за ним закріплено у системі є захищеними від несанкціонованого доступу, як і дані, якими користувач обмінюється із ботом. Систему авторизації також доповнює модель Admin, що містить у собі ідентифікатори всіх користувачів, що мають додаткові привілеї. Такими привілеями є описане далі резервне копіювання та відновлення.

Аспектом захисту, не менш важливим за захист від несанкціонованого доступу, є захист від катастроф, тобто непередбаченої втрати даних. Є кілька способів посилення такої безпеки, в тому числі до них відноситься і вибір СУБД, що буде відповідальна за дані системи. Таким чином обрано одну із найбезпечніших систем управління, що використовує всі сучасні техніки захисту від втрати даних. Однак, важливо також і вбезпечити дані від збою у системі керування, а тут найефективнішим способом є створення резервних копій різних видів.

У роботі реалізовано одразу два види резервних копій. Першою розглянемо резервну копію у SQL-файл, при зчитуванні якого сервер БД повністю відновить як базу даних, так і всі таблиці разом із інформацією, що була в них на момент створення копії. Для цього прямо із додатку викликається відповідна утиліта із набором налаштувань. Лістинг коду, що керує описаним процесом:

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

```

def cmd_backup(upd: Update, ctx: CallbackContext):
    with db_session.begin() as s:
        if s.get(Admin, upd.effective_user.id) is None:
            return
        message_data = ctx.bot.send_message(upd.effective_chat.id, 'Резервне
копіювання...')
        dt_start = datetime.datetime.now()
        backup_folder = BACKUP_FOLDER
        backup_dt = datetime.datetime.now().strftime("%d-%m-%Y_%H-%M")
        db_name = os.environ["DATABASE_URL"].split("/")[-1]
        backup_path = backup_folder +
f'/{db_name}_{backup_dt}_{generate_token(3)}_backup.sql'
        process = Popen(f'pg_dump.exe --file {backup_path} --host localhost --port 5432 '
f'--quote-all-identifiers --format=p --create --clean --
section=pre-data '
f'--section=data --section=post-data {db_name}'.split(' '),
stdout=subprocess.PIPE)
        process.wait()
        dt_end = datetime.datetime.now()
        ctx.bot.edit_message_text('Резервне копіювання успішно завершено!\n'
f'Шлях до копії: {backup_path}\n'
f'Часу знадобилося на копіювання: {dt_end -
dt_start}\n',
                                message_data.chat_id,
                                message_data.message_id)

```

Щоб відновити дані у додатку реалізовано і зворотний копіюванню процес, процес відновлення. Він використовує іншу утиліту, що виконує утворений підчас резервування файл та відновлює базу даних до того стану, у якому вона була на час резервування. Лістинг відповідного коду:

```

def cmd_restore(upd: Update, ctx: CallbackContext):
    with db_session.begin() as s:
        if s.get(Admin, upd.effective_user.id) is None:
            return
        message_data = ctx.bot.send_message(upd.effective_chat.id, 'Відновлення...')
        global db_engine
        db_engine.dispose()
        dt_start = datetime.datetime.now()
        backup_path = upd.message.text.split(' ')[1]
        process = Popen(f'psql -f {backup_path}'.split(' '), stdout=subprocess.PIPE)
        process.communicate()
        db_engine = create_engine()
        dt_end = datetime.datetime.now()
        ctx.bot.edit_message_text('Резервне копіювання успішно завершено!\n'
f'Часу знадобилося на відновлення: {dt_end -
dt_start}\n',
                                message_data.chat_id,
                                message_data.message_id)

```

Другим реалізованим способом є базова резервна копія, що допускає лише ручне відновлення. Така копія є повною копією всього сервера (що є її мінусом у випадку серверів із кількома, не пов'язаними між собою, базами даних), що використовує write-

ahead log для відновлення до будь-якої точки у часі між створенням копії та останньою командою, виконаною сервером. Лістинг коду, що виконує таку копію:

```
def cmd_base_backup(upd: Update, ctx: CallbackContext):
    with db_session.begin() as s:
        if s.get(Admin, upd.effective_user.id) is None:
            return
        message_data = ctx.bot.send_message(upd.effective_chat.id, 'Резервне
копіювання...')
        dt_start = datetime.datetime.now()
        backup_folder = BACKUP_FOLDER + '/BaseBackup'
        try:
            shutil.rmtree(backup_folder)
        except:
            pass
        process = Popen(f'pg_basebackup.exe -X stream -D {backup_folder}'.split(' '),
stdout=subprocess.PIPE)
        process.wait()
        dt_end = datetime.datetime.now()
        ctx.bot.edit_message_text('Резервне копіювання успішно завершено!\n'
                                f'Шлях до копії: {backup_folder}\n'
                                f'Часу знадобилося на копіювання: {dt_end -
dt_start}\n',
                                message_data.chat_id,
                                message_data.message_id)
```

4.2 Налаштування параметрів роботи PostgreSQL

Для коректної роботи сервера, взаємодії із ним та правильного утворення резервних копій різних типів проведено перелік налаштувань:

1. Налаштовано список доменів, що можуть підключатися до сервера (файл pg_hba.conf):

#	TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only					
local	all		all		scram-sha-256
# IPv4 local connections:					
host	all		all	127.0.0.1/32	scram-sha-256
# IPv6 local connections:					
host	all		all	:::1/128	scram-sha-256
# Allow replication connections from localhost, by a user with the					
# replication privilege.					
local	replication		all		scram-sha-256
host	replication		all	127.0.0.1/32	scram-sha-256
host	replication		all	:::1/128	scram-sha-256

2. Для уникнення необхідності авторизації налаштовані логін та пароль користувача за замовченням (використовується при операціях резервного копіювання та відновлення):

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

Name	Value
OS	Windows_NT
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC...
PGPASSWORD	76
PGUSER	P
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	AMD64 Family 23 Model 8 Stepping 2, AuthenticAMD
PROCESSOR_LEVEL	23

Рис. 4.1. Налаштування логіну та паролю за замовченням.

3. Щоб СУБД здійснювала журналювання із записом наперед налаштовано такі поля у файлі postgresql.conf:

```
wal_level = replica
archive_mode = on
archive_command = 'copy "%p" "D:\\PostgreSQL\\WALArchive\\%f"'
archive_timeout = 864000
restore_command = 'copy "D:\\PostgreSQL\\WALArchive\\%f" "%p"'
archive_cleanup_command = 'pg_archivecleanup "D:\\PostgreSQL\\WALArchive" %r'
```

Висновки до 4 розділу:

Було використано систему авторизації, надану Telegram API, що є основою для виконання даної роботи, для забезпечення захисту даних у базі, як від несанкціонованого перегляду, так і від несанкціонованої зміни.

Проведено налаштування системи керування базами даних таким чином, щоб користувач міг підключатися до сервера та щоб можна було коректно автоматизувати резервне копіювання як бази даних, так і сервера в цілому.

ВИСНОВКИ

Під час написання курсової роботи було отримано навички роботи з системою управління базами даних PostgreSQL у симбіозі із Python, використовуючи для їх поєднання SQLAlchemy та Psycopg3.

У першому розділі було поставлено задачу на основний та допоміжний функціонал у ході аналізу аналогічних програмних продуктів. Проаналізовано три СУБД, що підходили під критерії реляційності та відкритості коду, з них обрано ту, що найкраще себе зарекомендувала особливостями, що оцінювалися.

У другому розділі розроблено схему роботи додатку, в повній мірі описано функціональні особливості майбутнього додатку. Спроектовано моделі, з яких формувалися таблиці, що є необхідними для коректної та повної відповідності поставленим функціональним вимогам.

Результатом роботи над третім розділом став інтерфейс користувача, що надає йому всі необхідні можливості для зручної взаємодії із додатком та отримання візуальної інформації у вигляді звітів, саму ж звітність у вигляді графіків та агрегації наявних даних також написано саме під час виконання завдань цього розділу.

Четвертий розділ став наріжним каменем реалізації безпеки даних у системі як від несанкціонованого доступу, так і від непередбаченої катастрофи (тобто втрати) даних. У цьому ж розділі налаштовано сервер системи таким чином, щоб підключення, резервне копіювання та відновлення проходили бездоганно.

Продуктом цієї роботи став чат-бот на основі Telegram API, що реалізує всі необхідні у системі опитування речі та бази даних, яку він використовує для зберігання та захисту зібраної інформації.

		Каліновський Д. Ю.			ДУ «Житомирська політехніка».22.121.07.000 - ІПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

ЛІТЕРАТУРА

1. Advanced Python Programming: Build high performance, concurrent, and multi-threaded apps with Python using proven design patterns – 320 с. – (ISBN-10 – 1838551212)
2. SQLAlchemy 1.4 Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.sqlalchemy.org/en/14/index.html>
3. Python Telegram Bot's Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.python-telegram-bot.org/en/v13.12/>
4. GitHub of python-telegram-bot [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/python-telegram-bot/python-telegram-bot>
5. Matplotlib 3.5.2 Examples [Електронний ресурс] – Режим доступу до ресурсу: <https://matplotlib.org/stable/gallery/index>
6. PostgreSQL 14.4 Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/docs/14.4/index.html>
7. How to declare a variable in a PostgreSQL query – Stack Overflow [Електронний форум] / Stack Exchange Inc. – 2022 – Режим доступу до ресурсу: <https://stackoverflow.com/questions/1490942/how-to-declare-a-variable-in-a-postgresql-query>
8. How can I bind a list to a parameter in a custom query in SQLAlchemy? – Stack Overflow [Електронний форум] / Stack Exchange Inc. – 2022 – Режим доступу до ресурсу: <https://stackoverflow.com/questions/13190392/how-can-i-bind-a-list-to-a-parameter-in-a-custom-query-in-sqlalchemy>
9. Python Code Examples [Електронний ресурс] – Режим доступу до ресурсу: <https://www.programcreek.com/python/>
10. Методичні рекомендації для виконання курсової роботи Файл [Електронний ресурс] – Режим доступу до ресурсу: https://learn.ztu.edu.ua/pluginfile.php/8204/mod_resource/content/3/metod_kurs_full.pdf

ДОДАТКИ

Технічне завдання на роботу.

1. Загальне положення

1.1. Найменування програмного засобу

Найменування програмної системи: "Система опитувань на базі Telegram API".

1.2. Призначення розробки та область застосування

Призначенням розробки є проведення соціальних опитувань різного роду, від політичних аналізів до оцінки роботи персоналу, а також проведення тестувань у навчальному процесі будь-яких навчальних закладів із отриманням оцінки за спробу проходження, можливістю виправлення помилок у тестуванні та інструментами для аналізу отриманих даних.

1.3. Мета

База даних системи опитувань є зручним інструментом для збереження даних користувачів, створених опитувань, спроб проходження цих опитувань та швидкого автоматичного обрахування деяких показників у системі.

1.4. Найменування розробника та замовника.

Розробник даного продукту - студент групи ІПЗк-21-1 Каліновський Денис Юрійович (надалі «розробник»)

Замовник програмного продукту - кафедра інженерії програмного забезпечення державного університету «Житомирська політехніка» в межах виконання курсової роботи з дисципліни «Бази даних» - Чижмотря Олексій Володимирович.

2. Підстава для розробки

2.1. Документ на підставі якого ведеться розробка

Робота ведеться на підставі навчального плану за спеціальністю 121 «Інженерія програмного забезпечення».

3. Вимоги до програми

3.1. Вимоги до функціональних характеристик.

3.1.1. Загальні вимоги

Програмна система має забезпечувати:

- Можливість створення користувачем опитувань, що включають в себе категорії цих опитувань, їх запитання та відповіді на ці запитання;
- Можливість редагування створених опитувань, в тому числі налаштування їх приватності, активованості та редагування усіх їх властивостей;
- Проходження створених користувачами опитувань та тестувань з отриманням результатів у короткому та розгорнутому виглядах;
- Пошук опитувань серед публічних і активованих, що створені іншими користувачами (з можливістю їх проходження)
- Отримання статистики щодо проходження тестувань як зі сторони автора, так і зі сторони споживача;
- Можливість створення резервних копій та проведення відновлення даних.

3.1.2. Склад виконуваних функцій

Розробити базу даних, що використовується у Telegram API додатку, який підтримує виконання наступних операцій:

- Створення, перегляд та видалення опитувань, категорій, запитань та відповідей;
- Збереження спроб проходження опитувань із обрахунком оцінки за спробу, якщо вона потрібна;
- Резервне копіювання за допомогою офіційних додатків-інструментів;
- Агрегація даних у базі з отриманням зручного для перегляду набору даних;

3.1.3. Організація вхідних і вихідних даних

Вхідними даними є інформація про користувача та інформація, що вноситься в систему при додаванні або зміні значень полів в таблицях.

Введення оперативних даних повинно виконуватися з використанням діалогу із користувачем у Telegram з використанням текстових повідомлень та клавіатур повідомлень.

Введення даних виконується вручну користувачем за допомогою діалогів з чат-ботом та реалізованих можливостей взаємодії, після обробки дані зберігаються в БД.

3.1.4. Часові характеристики і розмір пам'яті, необхідної для роботи програми.

Час реакції програми на дії користувача (маніпуляції з пристроями введення даних) не повинен перевищувати 1 с.

Час виконання команд меню не більше 3 с.

Відображення масивів даних за запитами не більше 1 хвилини.

Доступність БД – 90% цілодобово.

Операції з'єднання з БД не більше 1 хвилини.

Обсяг оперативної пам'яті, необхідний для роботи програми не менше 1ГБ.

Дисковий простір, необхідний для збереження програми і файлів даних не менше 20 ГБ для серверної частини. Від користувача вимоги ставляться програмним забезпеченням, за допомогою якого він підключається до системи.

Інсталяційний пакет програми відсутній. Є лише пакет розгортання на сервері, що містить у собі пакет розгортання СУБД та не повинен займати більше 300 МБ.

3.2. Вимоги до надійності.

3.2.1. Вимоги до надійного функціонування

Програма повинна нормально функціонувати при безперебійній роботі ПК. Доступність БД 90% при одночасному доступі великої кількості користувачів (більше 1000).

3.2.2.Контроль вхідної і вихідної інформації

Для контролю коректності вхідної інформації та захисту від помилок оператора:

- Перевірка відповідності даних доменам інформаційних атрибутів;
- Використанням механізму автозаповнення та вибору за переліком для зв'язаних даних;
- Захист від помилок оператора (залипань, випадкових символів тощо). Визначені некоректні дії повинні супроводжуватись повідомленнями про помилку і блокуванням операцій оновлення даних. В системі має бути передбачений захист від загального блокування.

Для вихідної інформації передбачити:

- відсутність блокування даних через багатокористувальницький доступ;
- оновлення даних у відображених на екрані звітних формах за запитом користувача.
- повідомлення про використання режиму монопольного доступу адміністраторами БД.

3.3. Вимоги до інформаційної і програмної сумісності.

3.3.1. Вимоги до інформаційних структур на вході і виході

Формат відображення даних має дозволяти імпорт даних в додатки MS Office для редагування та перегляду.

3.3.2. Вимоги до методів рішення і мов програмування

Вибір методів рішення здійснюється розробником без узгодження з замовником. СУБД обирається у відповідності до характеристик визначених в п.3. З замовником погоджується вибір варіанту за вартісною ознакою необхідного серверного обладнання.

3.3.3. Вимоги до системи програмних засобів.

Вимоги до програмного забезпечення сервера: ОС – будь-яка версія ОС Windows або Linux не старше 2011 року останнього значного оновлення, СУБД визначається встановлюється та налаштовується розробником.

Вимоги до програмного забезпечення робочої станції встановлюються розробниками Telegram та програмних систем, використовуваних для доступу до Telegram.

3.5. Вимоги до складу і параметрів технічних засобів.

Вимоги до складу технічних засобів:

1. Сервер:

- сервер у базовій конфігурації із підтримкою RAID;
- зовнішній запам'ятовуючий пристрій для архівування даних;
- принтер для друку;
- засоби для під'єднання до локальної мережі.

2. Вимоги до робочої станції встановлюються розробниками Telegram та програмних систем, використовуваних для доступу до Telegram.

4. Вимоги до програмної документації

Програмна документація повинна включати наступні відомості:

1. «Керівництво користувача» надається користувачу під час користування програмною системою та є її невід'ємною частиною. Керівництво та його відображення користувачу реалізовано під час виконання програмного додатку.

2. «Керівництво адміністратора баз даних» складається з опису складу колекцій бази даних, правил доступу до них та порядку налаштування серверу. Під час оформлення пояснювальної записки до курсової роботи дані відомості містяться в 2, 3 та 4 розділах курсової роботи.

Найбільш важливі функції роботи з базою даних

Збереження ідентифікатора користувача при першій взаємодії

```
def cmd_start(upd: Update, ctx: CallbackContext):
    msg = f'Ви успішно авторизувалися у {BOT_NAME}.'
    try:
        with db_session.begin() as s:
            new_user = User(upd.effective_user.id)
            s.add(new_user)
    except IntegrityError as e:
        msg = f'Ви вже авторизовані у {BOT_NAME}!'

    ctx.bot.send_message(chat_id=upd.effective_chat.id, text=msg)
```

Збереження опитування

```
def conv_nq_success_end(upd: Update, ctx: CallbackContext):
    with db_session.begin() as s:
        user = s.get(User, upd.effective_user.id)

        quiz_ref = user.data['new_quiz']
        cats_ref = quiz_ref['categories']
        que_ref = quiz_ref['questions']

        if len(que_ref) < 1:
            ctx.bot.send_message(
                chat_id=upd.effective_chat.id,
                text=f'Опитування без запитань? Навіть звучить чудернацьки 😊\nБудь-
ласка, введіть запитання:')
            return NQ.QUE

        quiz = Quiz(quiz_ref['name'], user.id, quiz_ref['privacy'],
quiz_ref['is_statistical'])
        s.add(quiz)
        s.flush()

        for cat in cats_ref:
            kitten = QuizCategory(quiz.id, cat)
            s.add(kitten)

        for que in que_ref:
            question = QuizQuestion(quiz.id, que['question'], que['is_multi'])
            s.add(question)
            s.flush()

            for ans in que['right_answers']:
                answer = QuestionAnswer(question.id, ans, True)
                s.add(answer)
            for ans in que['wrong_answers']:
                answer = QuestionAnswer(question.id, ans, False)
                s.add(answer)

    ctx.bot \
        .send_message(
```

```

        chat_id=upd.effective_chat.id,
        text=f'Опитування успішно створено! Код опитування - {quiz.token}\nЙого
можна знайти за '
            f'{"назвою або " if quiz.is_public else ""}цим кодом у пошуку
(/search {quiz.token})'
            f'{" або /search " + quiz.name if quiz.is_public else ""}) або за
допомогою команди '
            f'/pass {quiz.token}. Цей код можна подивитися та змінити у меню
опитування.')
        user.remove_data('new_quiz')
        return ConversationHandler.END

```

Видалення відповіді, запитання, категорії та опитування

```

def answer_delete(upd: Update, ctx: CallbackContext):
    query = upd.callback_query
    action_split = query.data.split('.')
    msg = "Відповідь успішно видалено!"
    ret_value = MQ.ANSWER_MODE
    match action_split:
        case answer_id, 'yes':
            with db_session.begin() as s:
                answer = s.get(QuestionAnswer, answer_id)
                s.delete(answer)
        case answer_id, 'no':
            msg = "Видалення відповіді відмінено 🚫"
            ret_value = MQ.BACK_TO
    with db_session.begin() as s:
        user = s.get(User, upd.effective_user.id)
        query.edit_message_text(msg, reply_markup=InlineKeyboardMarkup([[
            InlineKeyboardButton('🏠 Повернутися до списку відповідей',
callback_data=f'{user.data["quiz_mode"] ["quiz_id"]}.answer_edit_mode')
        ]]))
    query.answer()
    return ret_value

def question_delete(upd: Update, ctx: CallbackContext):
    query = upd.callback_query
    action_split = query.data.split('.')
    msg = 'Запитання успішно видалено!'
    match action_split:
        case question_id, 'yes':
            with db_session.begin() as s:
                question = s.get(QuizQuestion, question_id)
                s.delete(question)
        case question_id, 'no':
            msg = "Видалення запитання відмінено 🚫"
    with db_session.begin() as s:
        user = s.get(User, upd.effective_user.id)
        query.edit_message_text(msg, reply_markup=InlineKeyboardMarkup([[
            InlineKeyboardButton('🏠 Повернутися до списку запитань',
callback_data=f'{user.data["quiz_mode"] ["quiz_id"]}.question_mode')
        ]]))
    query.answer()
    return MQ.BACK_TO

```

```

def cat_delete(upd: Update, ctx: CallbackContext):
    query = upd.callback_query
    action = query.data
    with db_session.begin() as s:
        user = s.get(User, upd.effective_user.id)
        match action:
            case 'quiz':
                quiz = s.get(Quiz, user.data['quiz_mode']['quiz_id'])
                query.edit_message_text(get_quiz_info(quiz),
                                         reply_markup=get_edit_quiz_keyboard(quiz))
                return MQ.EDIT

    query.edit_message_reply_markup(get_cat_keyboard(user.data['quiz_mode']['quiz_id'],
    action))
    query.answer()
    return MQ.CAT_MODE

def quiz_delete(upd: Update, ctx: CallbackContext):
    query = upd.callback_query
    action_split = query.data.split('.')
    msg = "Опитування успішно видалено!"
    match action_split:
        case quiz_id, 'yes':
            with db_session.begin() as s:
                quiz = s.get(Quiz, quiz_id)
                s.delete(quiz)
        case quiz_id, 'no':
            msg = "Видалення опитування відмінено 🚫"
    query.edit_message_text(msg, reply_markup=InlineKeyboardMarkup([[
        InlineKeyboardButton('🏠 Повернутися до списку опитувань',
callback_data=f'{upd.effective_user.id}.quiz_list'),
    ]]))
    query.answer()
    return MQ.BACK_TO

```

Редагування відповіді, запитання та опитування

```

def answer_edit(upd: Update, ctx: CallbackContext):
    answer_text = upd.message.text
    if RE_MED_TEXT.fullmatch(answer_text):
        with db_session.begin() as s:
            user = s.get(User, upd.effective_user.id)
            answer = s.get(QuestionAnswer, user.data['question_mode']['answer_id'])
            answer.answer = answer_text
            msg = ctx.bot.send_message(chat_id=upd.effective_chat.id,
                                         text="Відповідь успішно змінено!",
                                         reply_markup=InlineKeyboardMarkup(
                                             [[InlineKeyboardButton('Назад',
callback_data=f'{answer.id}.answer_edit_mode')]]))
            user.data['quiz_mode']['message_id'] = msg.message_id
            user.flag_data()
            return MQ.BACK_TO
    else:
        ctx.bot.send_message(chat_id=upd.effective_chat.id, text="Повідомлення містить

```

```

недопустимі символи або занадто "
                                                                    "довге : (")

def question_edit(upd: Update, ctx: CallbackContext):
    question_text = upd.message.text
    if RE_MED_TEXT.fullmatch(question_text):
        with db_session.begin() as s:
            user = s.get(User, upd.effective_user.id)
            question = s.get(QuizQuestion, user.data['question_mode']['question_id'])
            question.question = question_text
            msg = ctx.bot.send_message(
                chat_id=upd.effective_chat.id,
                text="Запитання успішно змінено!",
                reply_markup=InlineKeyboardMarkup(
                    [[InlineKeyboardButton('Назад',
callback_data=f'{question.id}.question_edit_mode')]])
            user.data['quiz_mode']['message_id'] = msg.message_id
            user.flag_data()
        else:
            ctx.bot.send_message(chat_id=upd.effective_chat.id, text="Повідомлення містить
недопустимі символи або занадто "
                                                                    "довге : (")

def quiz_edit(upd: Update, ctx: CallbackContext):
    query = upd.callback_query
    action_split = query.data.split('.')
    action_split[0] = int(action_split[0])
    query.answer()
    match action_split:
        case quiz_id, 'rename':
            with db_session.begin() as s:
                user = s.get(User, upd.effective_user.id)
                user.set_data('rename_quiz_id', quiz_id)

            query.edit_message_text(
                text=f'Введіть нову назву для свого опитування:',
                reply_markup=InlineKeyboardMarkup(
                    [[InlineKeyboardButton('Назад',
callback_data=f'{quiz_id}.quiz')]])
            return MQ.RENAME
        case quiz_id, 'privacy':
            with db_session.begin() as s:
                quiz = s.get(Quiz, quiz_id)
                quiz.is_public = not quiz.is_public
                s.flush()
            query.edit_message_reply_markup(get_edit_quiz_keyboard(quiz))
            return MQ.EDIT
        case quiz_id, 'availability':
            with db_session.begin() as s:
                quiz = s.get(Quiz, quiz_id)
                quiz.is_available = not quiz.is_available
                s.flush()
            query.edit_message_reply_markup(get_edit_quiz_keyboard(quiz))
            return MQ.EDIT
        case quiz_id, 'cat_mode':
            query.edit_message_text('Режим редагування категорій. Надішліть назву
категорії щоб її додати або '
                                                                    'натисніть на категорію під цим повідомленням щоб
її видалити.',
                                                                    reply_markup=get_cat_keyboard(quiz_id))

```

```

        with db_session.begin() as s:
            user = s.get(User, upd.effective_user.id)
            return MQ.CAT_MODE
        # case quiz_id, 'show_questions': with db_session.begin() as s: quiz =
s.get(Quiz, quiz_id)
        # ctx.bot.send_message( chat_id=upd.effective_chat.id,
text=str('\n\n'.join([f'{x + 1}. ' + str(
        # quiz.questions[x]) for x in range(len(quiz.questions))]))))
        case quiz_id, 'regenerate_token':
            with db_session.begin() as s:
                quiz = s.query(Quiz).filter_by(quiz_id=quiz_id).one_or_none()
                quiz.regenerate_token()
                query.edit_message_text(get_quiz_info(quiz_id),
reply_markup=get_edit_quiz_keyboard(quiz_id))
            return MQ.EDIT
        case quiz_id, 'show_stats':
            query.edit_message_text('Оберіть тип статистики, яку хочете переглянути:',
                                reply_markup=get_stats_keyboard())

            return MQ.STATS
        case quiz_id, 'question mode':
            with db_session.begin() as s:
                user = s.get(User, upd.effective_user.id)
                user.set_data('question_mode', {
                    'page': 0,
                })
            query.edit_message_text('Оберіть запитання, яке хочете змінити або введіть
нове, щоб почати створення '
                                'нового запитання:',

reply_markup=get_question_mode_keyboard(upd.effective_user.id))
            return MQ.QUESTION_MODE
        case quiz_id, 'delete':
            with db_session.begin() as s:
                quiz = s.get(Quiz, quiz_id)
                query.edit_message_text(f'Ви впевнені, що хочете видалити опитування
"{quiz.name}"?',
                                reply_markup=InlineKeyboardMarkup(
                                    [[InlineKeyboardButton('Так',
callback_data=f'{quiz_id}.yes'),
                                    InlineKeyboardButton('Hi',
callback_data=f'{quiz_id}.no')]]))
            return MQ.DELETE
        case quiz_id, 'quiz_list':
            query.edit_message_text('Оберіть опитування:',

reply_markup=get_all_quizzes_keyboard(upd.effective_user.id))
            return MQ.SHOW

```

Обробка вводу користувача під час проходження опитування

```

def answer_callback(upd: Update, ctx: CallbackContext):
    query = upd.callback_query
    action = query.data
    query.answer()
    if action.isnumeric():
        action = int(action)
        with db_session.begin() as s:
            user = s.get(User, upd.effective_user.id)
            is_multi = user.data['pass']['is_multi']

```

```

        selection = user.data['pass']['selection']
        if action in selection:
            selection.remove(action)
        else:
            if not is_multi:
                selection.clear()
            selection.append(action)
        user.flag_data()
        query.edit_message_reply_markup(get_current_markup(upd.effective_user.id))
    else:
        user_id = upd.effective_user.id
        if not update_question(query.edit_message_text, user_id, action):
            with db_session.begin() as s:
                attempt_id = session_to_attempt(user_id)
                attempt = s.get(Aattempt, attempt_id)
                quiz = s.get(Quiz, attempt.quiz_id)
                if not quiz.is_statistical:
                    retry_number = s.query(Aattempt).filter_by(user_id=user_id,
quiz_id=attempt.quiz_id).count()
                    query.edit_message_text(f'Обрахування результатів тестування...')
                    query.delete_message()
                    ctx.bot.send_photo(
                        upd.effective_chat.id,
                        get_answer_distribution(attempt_id),
                        f'Результати тестування:\n'
                        f'Унікальний код: {attempt.uuid}\n'
                        f'Назва: {quiz.name}\n'
                        f'Код опитування: {quiz.token}\n'
                        f'Час проходження: {(attempt.finished_on -
attempt.started_on)}\n'
                        f'Спроба №: {retry_number}\n'
                        f'Оцінка: {attempt.mark}/100\n'
                        f'Переглянути цю спробу можна надіславши боту команду:\n'
                        f'/show {attempt.uuid}')
                else:
                    query.edit_message_text('Опитування пройдено, ваші відповіді
успішно записано. Дякуємо за участь!')
            return ConversationHandler.END

```

Обробка даних проходження тестування та формування із них графіка

```

def get_answer_distribution(attempt_id: int):
    colors = []
    values = []

    with db_session.begin() as s:
        attempt = s.get(Aattempt, attempt_id)
        right_count = s.query(AattemptAnswer).filter(and_(
            AattemptAnswer.attempt_id == attempt_id,
            AattemptAnswer.mark == Decimal('1'))).count()
        partially_right_count = s.query(AattemptAnswer).filter(and_(
            AattemptAnswer.attempt_id == attempt_id,
            and_(AattemptAnswer.mark > Decimal('0'), AattemptAnswer.mark <
Decimal('1')))).count()
        wrong_count = s.query(AattemptAnswer).filter(and_(
            AattemptAnswer.attempt_id == attempt_id,
            AattemptAnswer.mark == Decimal('0'))).count()
        unanswered_count =
s.query(QuizQuestion).filter_by(quiz_id=attempt.quiz_id).count() - \

```



```

(right_count + wrong_count + partially_right_count)

if unanswered_count > 0:
    values.append(unanswered_count)
    colors.append('#AAAAAA')
if wrong_count > 0:
    values.append(wrong_count)
    colors.append('#FF0000')
if partially_right_count > 0:
    values.append(partially_right_count)
    colors.append('#FF9900')
if right_count > 0:
    values.append(right_count)
    colors.append('#00FF00')

data = np.array([values])
data_cum = data.cumsum(axis=1)

fig, ax = plt.subplots(figsize=(50, 3))
ax.invert_yaxis()
ax.xaxis.set_visible(False)
ax.set_xlim(0, np.sum(data, axis=1).max())

for i, (color) in enumerate(colors):
    widths = data[:, i]
    starts = data_cum[:, i] - widths
    rects = ax.barh([''], widths, left=starts, height=0.5, color=color)

    ax.bar_label(rects, label_type='center', color='black', fontsize=100)

out = io.BytesIO()
FigureCanvas(fig).print_png(out)
out.seek(0)
return out

```

Обробка даних проходжень опитування / тестування та формування відповідних графіків

```

def send_question_stats(upd, ctx, quiz_id, period, part):
    images = []
    with db_session.begin() as s:
        titles = [x[0] for x in
s.query(QuizQuestion.question).filter_by(quiz_id=quiz_id).all()]
        question_ids = s.query(QuizQuestion.id).filter_by(quiz_id=quiz_id).all()

        for que_id_index in range(len(question_ids)):
            que_id = question_ids[que_id_index][0]
            title = titles[que_id_index]
            labels = [x[0] for x in
s.query(QuestionAnswer.answer).filter_by(question_id=que_id).all()]
            answer_ids = [x[0] for x in
s.query(QuestionAnswer.id).filter_by(question_id=que_id).all()]
            stmt = text(
                "SELECT question_answer.answer, COUNT(question_answer.id) \n"
                "FROM question_answer \n"
                "INNER JOIN attempt_answer aa ON question_answer.id=ANY(aa.answer_ids)
\n"
                "INNER JOIN attempt att ON att.id = aa.attempt_id \n" +

```

```

        (f"AND DATE_PART('day', DATE_TRUNC(:period, NOW())) -
DATE_TRUNC(:period, att.started_on)) <= {time_parts_day_eq[period]} \n" if period !=
'all' else "" ) +
        "WHERE question_answer.id = ANY(:ans_ids)\n"
        "GROUP BY question_answer.id;")

counts = s.execute(stmt, {'ans_ids': answer_ids, 'period': period}).all()
if len(counts) < 1:
    continue

local_palette = palette[:len(counts)]
while len(local_palette) > len(counts):
    local_palette.extend(palette[:len(counts) - len(local_palette)])

fig, ax = plt.subplots()
ax.pie([x[1] for x in counts],
        labels=[(x[0] if len(x[0]) < 40 else x[0][:37] + '...') for x in
counts],
        radius=5,
        center=(10, 10),
        autopct='%1.1f%%',
        explode=[.08 for x in range(len(counts))],
        colors=local_palette,
        # wedgeprops={"linewidth": 1, "edgecolor": "white"},
        shadow=True)
ax.axis('equal')
plt.text(-2.3, 16.5, str(que_id_index + 1) + '. ' + (title if len(title) <
50 else title[:47] + '...'), fontsize=18, fontweight='bold')
# plt.tight_layout()
fig.set_size_inches(12, 7)
out = io.BytesIO()
FigureCanvas(fig).print_png(out)
out.seek(0)
images.append(InputMediaPhoto(out))
if len(images) > 1:
    while len(images) > 1:
        ctx.bot.send_media_group(upd.effective_chat.id, images[:10])
        images = images[10:]
if len(images) == 1:
    ctx.bot.send_photo(upd.effective_chat.id, images[0].media)

def send_attempt_count_stats(upd, ctx, quiz_id, period, part):
    with db_session.begin() as s:
        stats = s.execute(text(
            f""""SELECT DATE_TRUNC(:part, att.started_on), COUNT(*) as "Number of
attempts"
            FROM attempt att
            WHERE :quiz_id = att.quiz_id """" +
            (f"AND DATE_PART('day', DATE_TRUNC(:period, NOW())) - DATE_TRUNC(:period,
att.started_on)) <= {time_parts_day_eq[period]}" if period != 'all' else "" ) +
            """" GROUP BY DATE_TRUNC(:part, att.started_on)
            ORDER BY DATE_TRUNC(:part, att.started_on);
            """"), {'quiz_id': quiz_id, 'part': part, 'period': period}).all()

    fig, ax = plt.subplots()

    ax.plot(
        [x[0].strftime('%d.%m\n%Y') for x in stats], [x[1] for x in stats])

    ax.set(ylabel='Кількість проходжень')

```

```

photo = io.BytesIO()
FigureCanvas(fig).print_png(photo)
photo.seek(0)

ctx.bot.send_photo(upd.effective_chat.id, photo)

```

Трігери обрахунку оцінки за тестування та переобрахунку після зміни відповідей

```

from db.engine import db_session
from sqlalchemy import text

stmt = \
    """DROP FUNCTION IF EXISTS update_mark_for_attempt_answer CASCADE;
DROP FUNCTION IF EXISTS update_mark_for_attempt CASCADE;
DROP FUNCTION IF EXISTS get_mark_for_attempt CASCADE;
DROP FUNCTION IF EXISTS update_mark_for_related_attempts CASCADE;
DROP FUNCTION IF EXISTS get_mark_for_attempt_answer CASCADE;

CREATE FUNCTION get_mark_for_attempt(var_attempt_id bigint) RETURNS numeric AS $$
DECLARE
    var_quiz_id integer;
    var_question_count integer;
    var_mark numeric;
BEGIN
-- Get quiz id
SELECT attempt.quiz_id INTO var_quiz_id
FROM attempt
WHERE attempt.id = var_attempt_id;

-- Get question count
SELECT COUNT(*) INTO var_question_count
FROM quiz_question qq
WHERE qq.quiz_id = var_quiz_id;

-- Calculate the mark
SELECT 100 * ROUND(SUM(aa.mark) / var_question_count, 2) INTO var_mark
FROM attempt_answer aa
WHERE aa.attempt_id = var_attempt_id;

RETURN var_mark;
END $$ LANGUAGE 'plpgsql';

CREATE FUNCTION get_mark_for_attempt_answer(var_attempt_answer_id bigint) RETURNS
numeric AS $$
DECLARE
    var_quiz_id integer;
    var_question_id bigint;
    var_question_count integer;
    var_weight numeric;
    var_right_answer_count integer;
    var_raw_mark numeric;
    var_mark numeric;
BEGIN
-- Get question id
SELECT aa.question_id INTO var_question_id FROM attempt_answer aa WHERE aa.id =

```

```

var_attempt_answer_id;

-- Calculate answer weight
SELECT 1. / COUNT(*) INTO var_weight
FROM quiz_question
INNER JOIN question_answer qa ON question_id = quiz_question.id
WHERE question_id = var_question_id AND qa.is_right = true
GROUP BY quiz_question.id;

-- Get answers for a question
DROP TABLE IF EXISTS tmp_answers;
CREATE TEMP TABLE tmp_answers AS
SELECT qa.is_right
FROM question_answer qa
INNER JOIN attempt_answer aa ON qa.id = ANY(aa.answer_ids)
WHERE aa.id = var_attempt_answer_id;

-- Count right answers
SELECT COUNT(*) INTO var_right_answer_count
FROM tmp_answers
WHERE is_right = true;

-- Calculate raw mark (can be negative)
SELECT ROUND((var_right_answer_count * var_weight) - ((COUNT(*) -
var_right_answer_count) * var_weight), 2) INTO var_raw_mark
FROM tmp_answers;

-- If mark < 0, mark = 0
SELECT COALESCE(NULLIF (ABS(var_raw_mark), -var_raw_mark), 0) INTO var_mark;

RETURN var_mark;
END $$ LANGUAGE 'plpgsql';

```

```

CREATE FUNCTION update_mark_for_attempt() RETURNS TRIGGER AS $$
DECLARE
    var_mark numeric;
BEGIN
-- Get the mark
SELECT get_mark_for_attempt(OLD.attempt_id) INTO var_mark;

-- Update mark of given attempt
UPDATE attempt
SET mark = var_mark
WHERE attempt.id = OLD.attempt_id;

RETURN OLD;
END $$ LANGUAGE 'plpgsql';

```

```

CREATE FUNCTION update_mark_for_attempt_answer() RETURNS TRIGGER AS $$
DECLARE
    var_mark numeric;
    var_row RECORD;
BEGIN
-- Recalculate mark for attempt_answer
SELECT get_mark_for_attempt_answer(NEW.id) INTO var_mark;

-- Update attempt answers' mark

```

```

    IF NEW.mark != var_mark THEN
        UPDATE attempt_answer aa
        SET mark = var_mark
        WHERE id = NEW.id;
    END IF;
-- Recalculate mark for attempt
SELECT get_mark_for_attempt(NEW.attempt_id) INTO var_mark;

DROP TABLE IF EXISTS temptemp;
CREATE TABLE temptemp AS SELECT * FROM get_mark_for_attempt(NEW.attempt_id);

-- Update attempts' mark
UPDATE attempt
SET mark = var_mark
WHERE NEW.attempt_id = attempt.id;
RETURN NEW;
END $$ LANGUAGE 'plpgsql';

-- TODO: never executed
CREATE FUNCTION update_mark_for_related_attempts() RETURNS TRIGGER AS $$
DECLARE
    var_quiz_id int;
    var_weight numeric;
    var_right_answer_count integer;
    var_mark numeric;
    temp_row RECORD;

    var_question_id bigint;
BEGIN
    IF NEW.question_id IS NULL THEN
        var_question_id = OLD.question_id;
    ELSE
        var_question_id = NEW.question_id;
    END IF;

-- Get quiz id
SELECT quiz_id INTO var_quiz_id FROM quiz_question qq WHERE qq.id =
var_question_id;

FOR temp_row IN
    SELECT * FROM attempt_answer aa
    WHERE aa.question_id = var_question_id
LOOP
    SELECT get_mark_for_attempt_answer(temp_row.id) INTO var_mark;
    UPDATE attempt_answer aa
    SET mark = var_mark
    WHERE aa.id = temp_row.id;
END LOOP;

FOR temp_row IN
    SELECT * FROM attempt att
    WHERE att.quiz_id = var_quiz_id
LOOP
    SELECT get_mark_for_attempt(temp_row.id) INTO var_mark;
    UPDATE attempt att
    SET mark = var_mark
    WHERE att.id = temp_row.id;
END LOOP;

RETURN NEW;
END $$ LANGUAGE 'plpgsql';

```

```

CREATE TRIGGER attempt_answer_insert_trigger
  AFTER INSERT ON attempt_answer
  FOR EACH ROW
--      WHEN (OLD.answer_ids IS DISTINCT FROM NEW.answer_ids)
EXECUTE FUNCTION update_mark_for_attempt_answer();

```

```

CREATE TRIGGER attempt_answer_update_trigger
  AFTER UPDATE ON attempt_answer
  FOR EACH ROW
--      WHEN (OLD.answer_ids IS DISTINCT FROM NEW.answer_ids)
  WHEN (OLD.mark IS NOT DISTINCT FROM NEW.mark)
EXECUTE FUNCTION update_mark_for_attempt_answer();

```

```

CREATE TRIGGER attempt_answer_delete_trigger
  AFTER DELETE ON attempt_answer
  FOR EACH ROW
EXECUTE FUNCTION update_mark_for_attempt();

```

```

CREATE TRIGGER question_answer_insert_trigger
  AFTER INSERT ON question_answer
  FOR EACH ROW
EXECUTE FUNCTION update_mark_for_related_attempts();

```

```

CREATE TRIGGER question_answer_update_trigger
  AFTER UPDATE ON question_answer
  FOR EACH ROW
  WHEN (OLD.is_right IS DISTINCT FROM NEW.is_right)
EXECUTE FUNCTION update_mark_for_related_attempts();

```

```

CREATE TRIGGER question_answer_delete_trigger
  AFTER DELETE ON question_answer
  FOR EACH ROW
EXECUTE FUNCTION update_mark_for_related_attempts();""

```

```

with db_session.begin() as s:
    s.execute(text(stmt))

```