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 ІI курсу групи ІПЗк-21-1

спеціальності 121 «Інженерія програмного забезпечення»

Каліновського Дениса Юрійовича

(прізвище, ім’я та по-батькові)

Керівник ст. викладач кафедри ІПЗ Олексій ЧИЖМОТРЯ

Дата захисту: " " липня 2022 р.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Інна СУГОНЯК

(підпис) (ім’я та прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ольга КОРОТУН

(підпис) (ім’я та прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Світлана КРАВЧЕНКО

(підпис) (ім’я та прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Олексій ЧИЖМОТРЯ

(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.

1. Перелік графічного матеріалу(з точним зазначенням обов’язкових креслень)

1. Презентація до КП

2. Посилання на репозиторій: <https://gitlab.com/dk000/quizzy>

1. 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 |

1. 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** \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Денис КАЛІНОВСЬКИЙ

(підпис) (прізвище та ініціали)

**Керівник роботи** \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Олексій ЧИЖМОТРЯ

(підпис) (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.

ЗМІСТ

[ВСТУП 6](#_Toc109304610)

[РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТОКІВ ТА ОСОБЛИВОСТЕЙ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ 8](#_Toc109304611)

[1.1 Аналіз інформаційних потреб та визначення предметної області дослідження 8](#_Toc109304612)

[1.2 Обґрунтування вибору засобів реалізації 12](#_Toc109304613)

[Висновки до першого розділу 13](#_Toc109304614)

[РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ СИСТЕМИ ОПИТУВАНЬ 14](#_Toc109304615)

[2.1 Аналіз інформаційних процесів системи опитувань 14](#_Toc109304616)

[2.2 Проектування структури бази даних системи опитувань 16](#_Toc109304617)

[2.3 Обробка даних системи 21](#_Toc109304618)

[Висновки до другого розділу 23](#_Toc109304619)

[РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПІДСИСТЕМИ ОБРОБКИ ДАНИХ СИСТЕМИ ОПИТУВАНЬ 24](#_Toc109304620)

[3.1 Проектування інтерфейсу обробки даних 24](#_Toc109304621)

[3.2 Реалізація операцій обробки даних чат-ботом 28](#_Toc109304622)

[3.3 Реалізація організації звітності системи 34](#_Toc109304623)

[Висновки до 3 розділу: 37](#_Toc109304624)

[РОЗДІЛ 4. АДМІНІСТРУВАННЯ БАЗ ДАНИХ 38](#_Toc109304625)

[4.1 Розробка заходів захисту інформації в БД 38](#_Toc109304626)

[4.2 Налаштування параметрів роботи PostgreSQL 40](#_Toc109304627)

[Висновки до 4 розділу: 41](#_Toc109304628)

[ВИСНОВКИ 42](#_Toc109304629)

[ЛІТЕРАТУРА 43](#_Toc109304630)

[ДОДАТКИ 44](#_Toc109304631)

# 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** єрозробка 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.

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.

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

## 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 запитами, 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 гармонує 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.

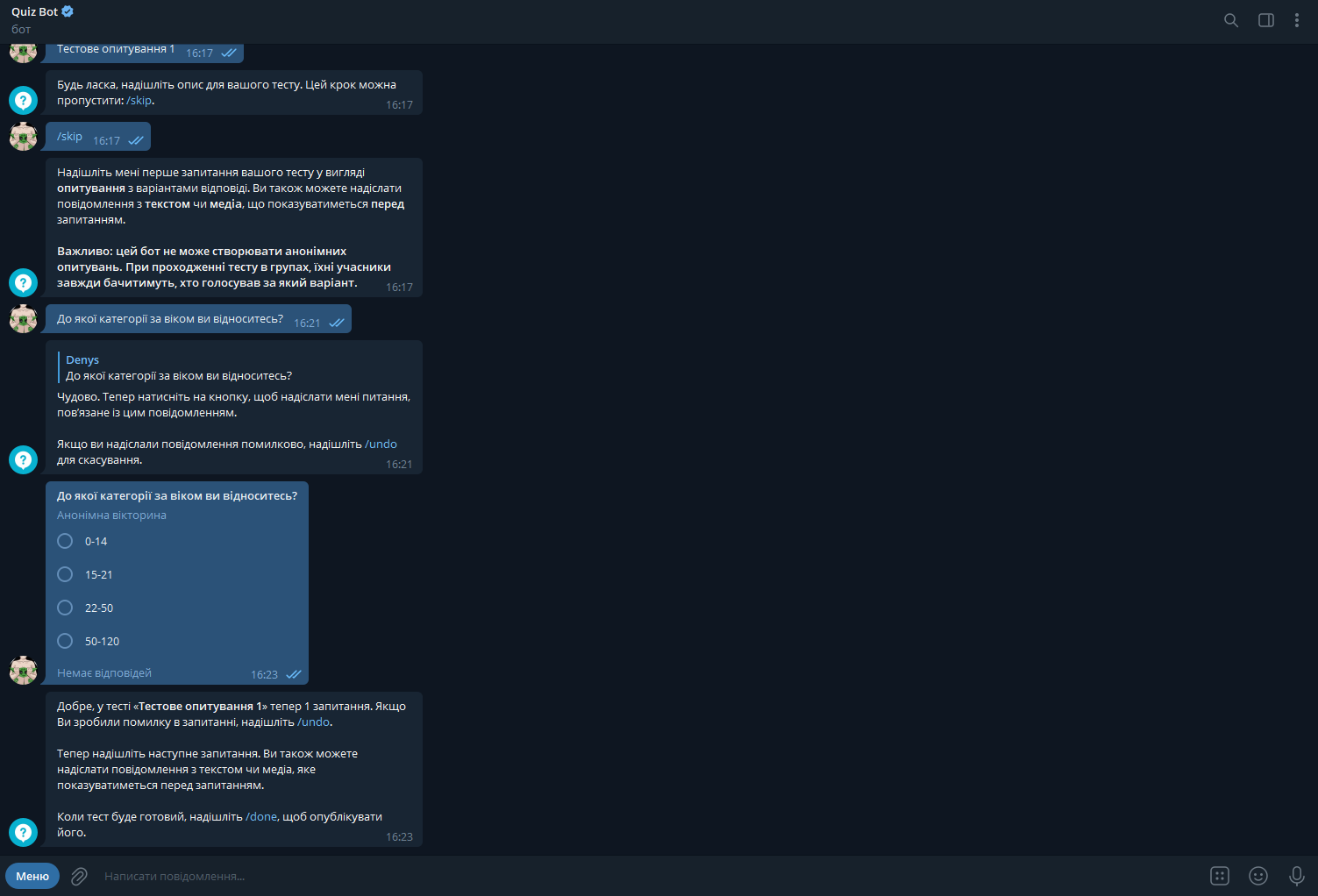
На сьогоднішній день сервісів для проведення опитувань є не дуже багато, однак існуючі аналоги є дуже якісними та мають широкий набір функцій. Перед тим як розглянути їх варто зауважити, що всіх них є великий мінус: вони мають закритий джерельний код, тобто використовуючи їх не можна гарантувати об’єктивності результатів опитувань, на відміну від рішення, що буде створено як продукт цієї роботи.

Розглянемо три найближчих найпопулярніших та найближчих за тематикою додатки:

1. Telegram-bot Quiz Bot.

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

Зовнішній вигляд чат-бота:

  
Рис. 1.1. Quiz Bot.

Переваги:

1. Базується на Telegram API, яким користуються мільйони людей, тобто не потребує ще однієї реєстрації від користувача.
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.
3. Платформа Typeform

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

Зовнішній вигляд при створенні опитування:

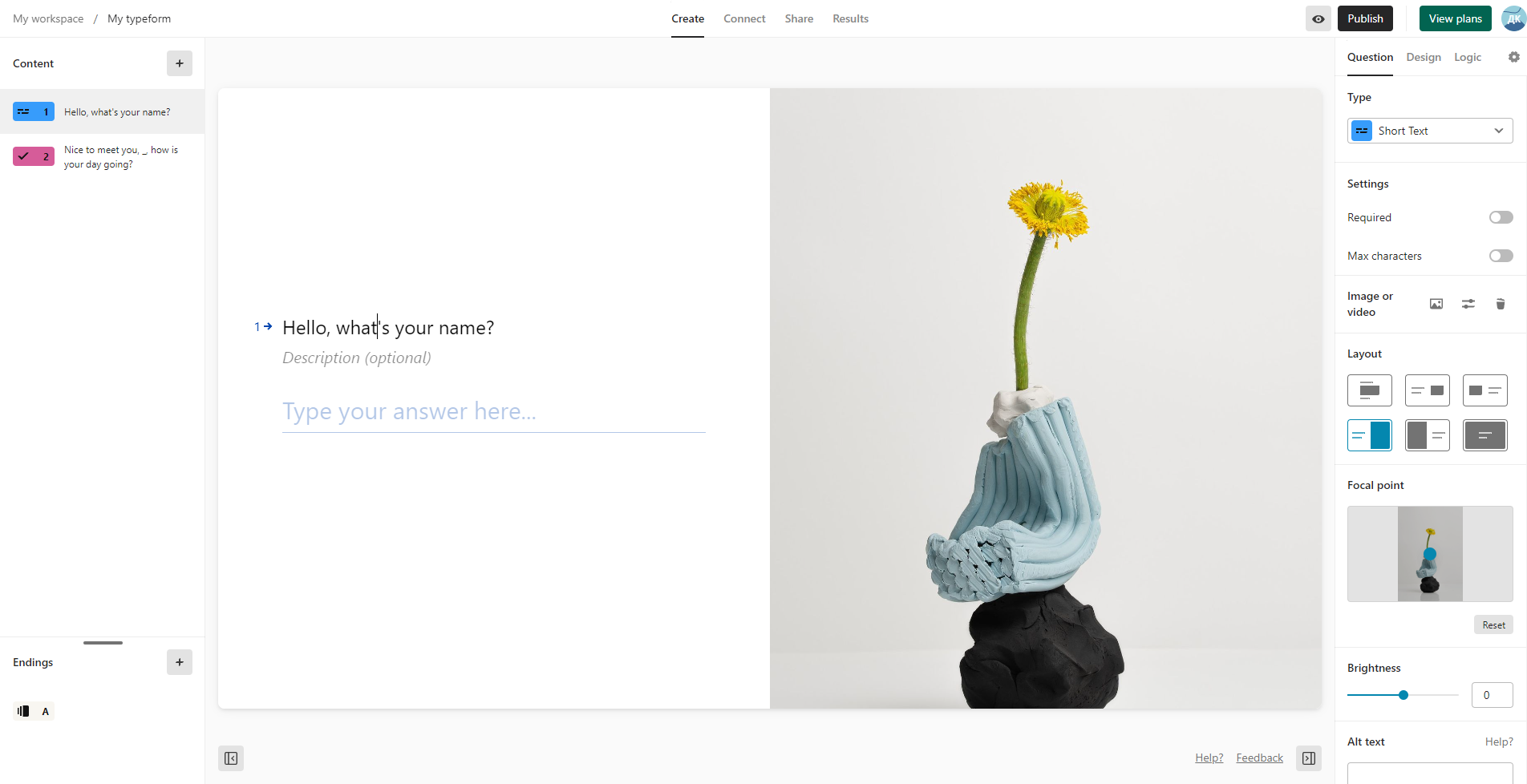


Рис. 1.2. Створення опитування у Typeform.

Переваги:

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.
5. Веб-сайт Quiz Maker.

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

Зовнішній вигляд:

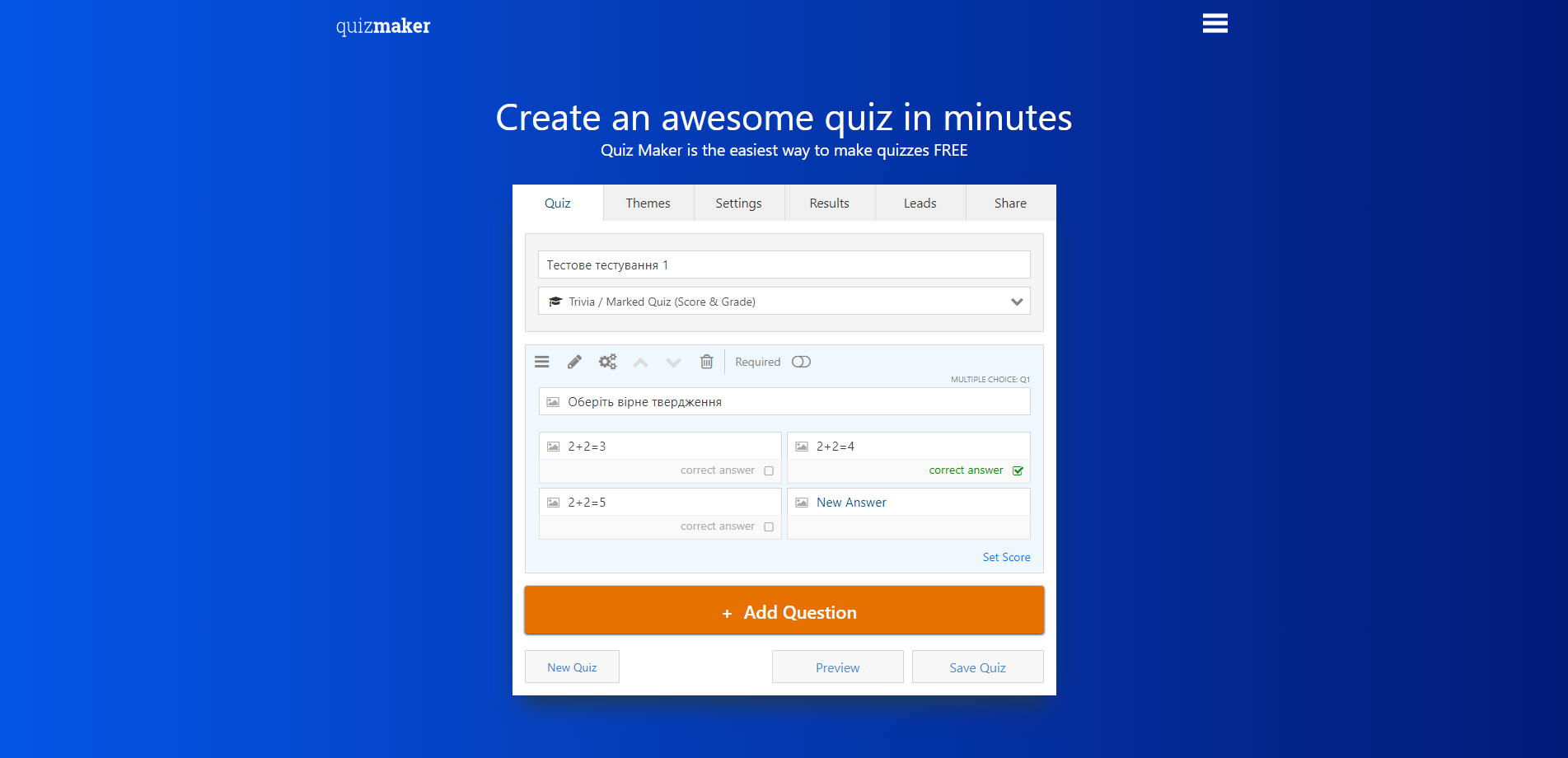


Рис. 1.3. Сайт Quiz Maker.

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.

## 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 розроблює 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 підчас розробки. Проаналізовано 3 популярні реляційні СУБД із відкритим джерельним кодом і обрано ту, що виявилася найкращою та що має найбільший об’єм функціональних можливостей.

# РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ СИСТЕМИ ОПИТУВАНЬ

## 2.1 Аналіз інформаційних процесів системи опитувань

На етапі проектування складено алгоритм роботи системи опитувань в рамках Telegram чат-бота (рис.2.1).

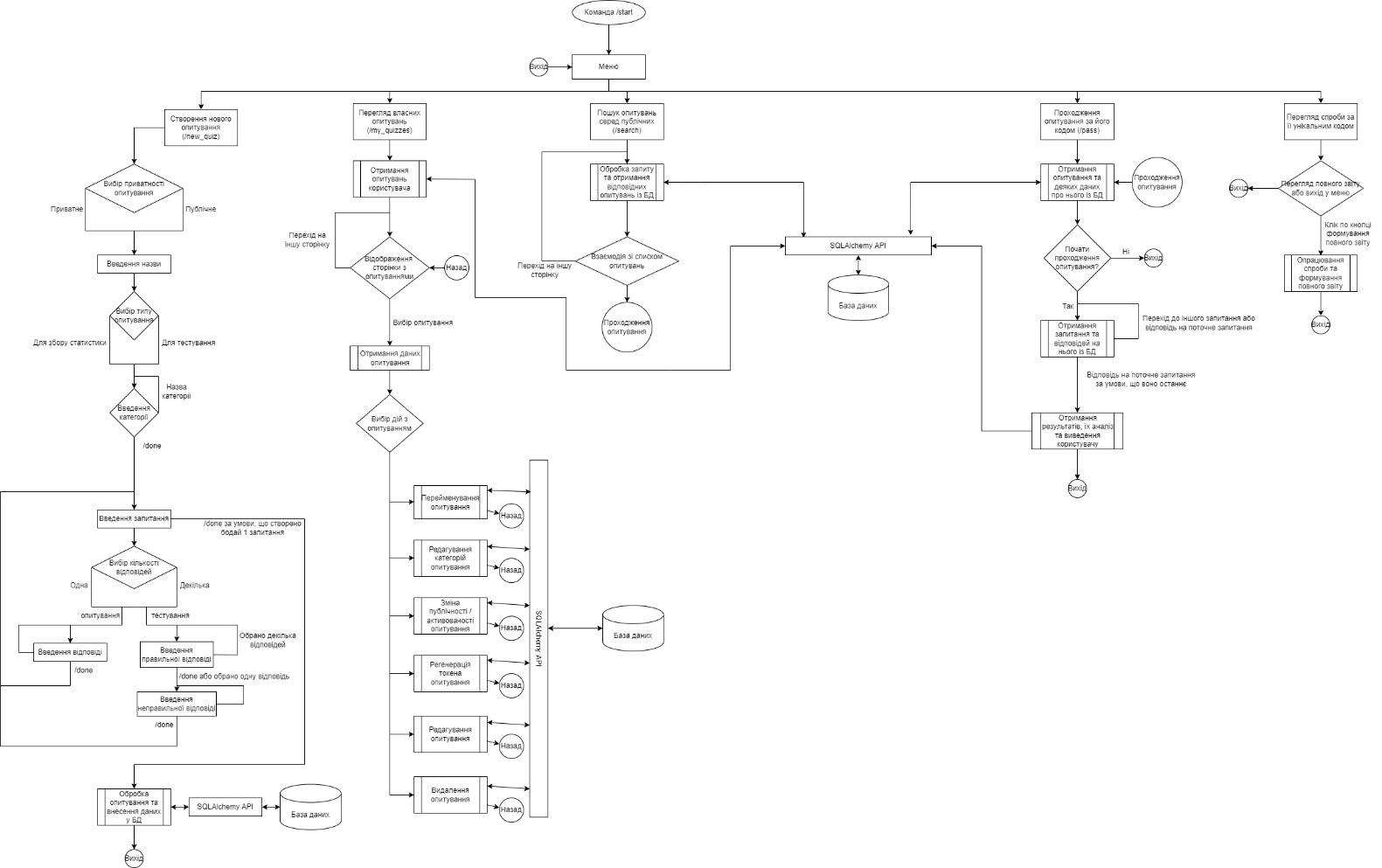


Рис. 2.1. Схема принципу роботи із середнім рівнем абстракції.

Після переходу у месенджері до діалогу з ботом з’являється пропозиція авторизуватися (надіслати команду “/start”). Після авторизації стають доступними всі аспекти роботи з додатком окрім резервного копіювання та відновлення, що доступні лише користувачам-адміністраторам.

У додатку реалізовано перелік функцій у вигляді «розмов» з ботом, що починаються із відповідних команд:

1. /new\_quiz – створення нового опитування:
   1. Користувач вводить назву опитування;
   2. Обирає тип опитування (опитування чи тестування);
   3. Вводить категорії опитування;
   4. Вводить запитання або за бажанням переходить до останнього пункту якщо створено бодай одне запитання;
   5. Обирає кількість відповідей (одна або декілька);
   6. Вводить вірні відповіді (якщо створюється статистичне опитування то користувач вводить відповіді без прив’язки до правильності);
   7. Вводить не вірні відповіді (цей етап пропускається у випадку із статистичним опитуванням);
   8. Переходить до пункту d або до наступного;
   9. Користувач отримує інформацію про своє опитування (назва, код і тому подібне).
2. /my\_quizzes – перегляд користувачем власних опитувань:
   1. Вибір опитування;
   2. Зміна будь-яких даних опитування, що не є незмінними (тип опитування), видалення опитування або перехід до пункту а;
   3. Перехід до списку запитань;
   4. Зміна будь-яких даних запитань, що не є незмінними (тип запитання), видалення запитання або перехід до пункту c;
   5. Перехід до списку відповідей;
   6. Зміна будь-яких даних запитання, видалення відповіді або перехід до пункту е.
3. /pass – проходження опитування за його кодом:
   1. Користувач обирає, чи хоче проходити це опитування;
   2. 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;
   3. 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:
   1. 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.a.
5. /show – перегляд спроби за її унікальним кодом:
   1. 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:
   1. 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:
   1. 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:
   1. 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.

В рамках виконання роботи створено наступні таблиці:

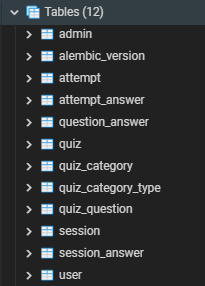


Рис. 2.2. Створені таблиці.

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:

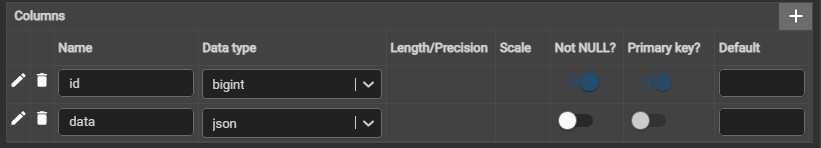


Рис. 2.3. Таблиця user.

Таблиця, що є результатом обробки моделі Quiz:

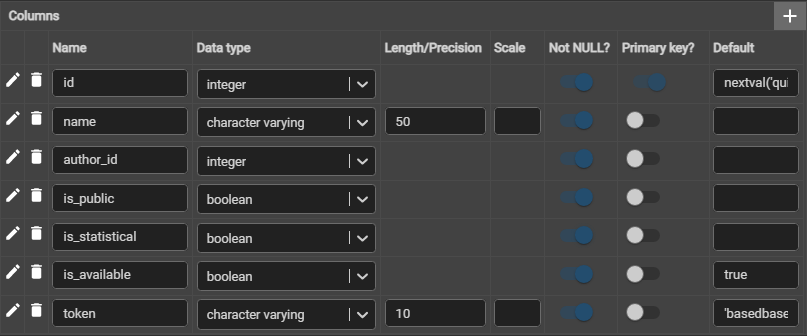


Рис. 2.4. Таблиця quiz.

Таблиця, що є результатом обробки моделі QuizQuestion:

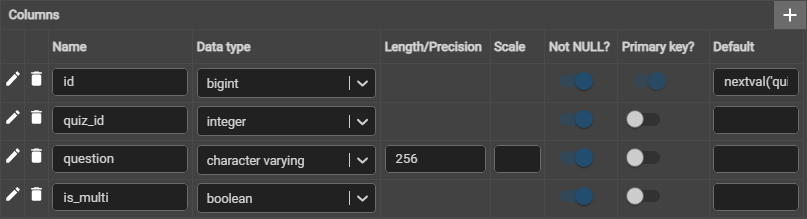


Рис. 2.5. Таблиця quiz\_question.

Таблиця, що є результатом обробки моделі QuizCategoryType:

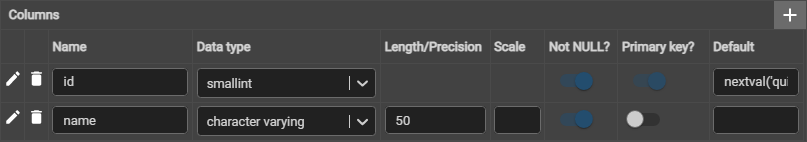


Рис. 2.6. Таблиця quiz\_category\_type.

Таблиця, що є результатом обробки моделі QuizCategory:

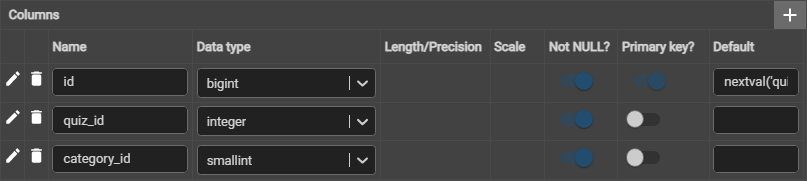


Рис. 2.7. Таблиця quiz\_category.

Таблиця, що є результатом обробки моделі QuestionAnswer:

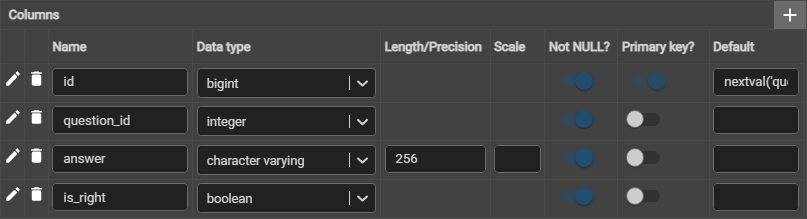


Рис. 2.8. Таблиця question\_answer.

Таблиця, що є результатом обробки моделі Admin:

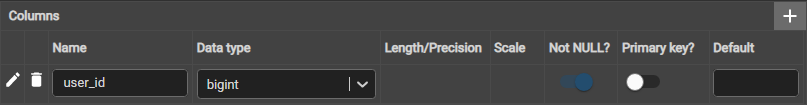


Рис. 2.9. Таблиця admin.

Таблиця, що є результатом обробки моделі Session:

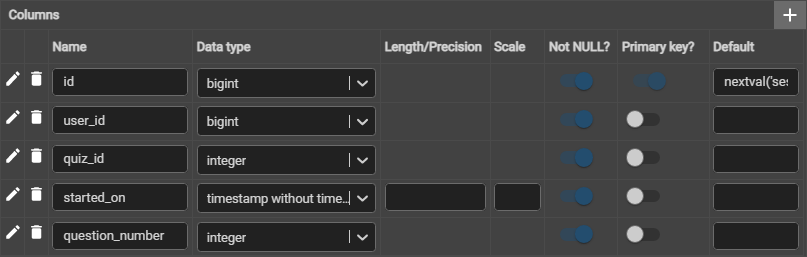


Рис. 2.10. Таблиця session.

Таблиця, що є результатом обробки моделі SessionAnswer:

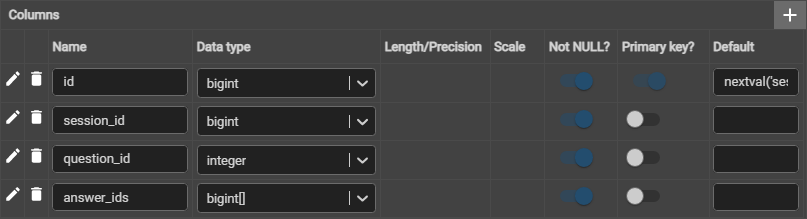


Рис. 2.11. Таблиця session\_answer.

Таблиця, що є результатом обробки моделі Attempt:

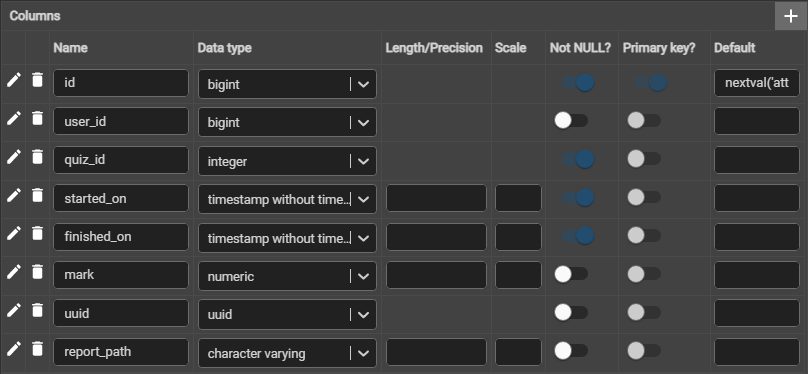


Рис. 2.12. Таблиця attempt.

Таблиця, що є результатом обробки моделі AttemptAnswer:

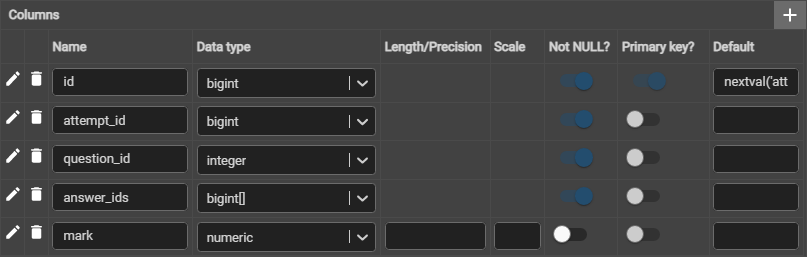


Рис. 2.13. Таблиця attempt\_answer.

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:

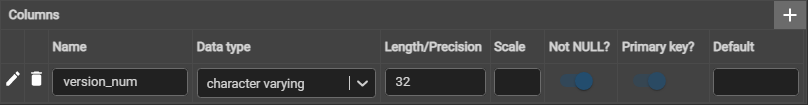


Рис. 2.14. Таблиця alembic\_version.

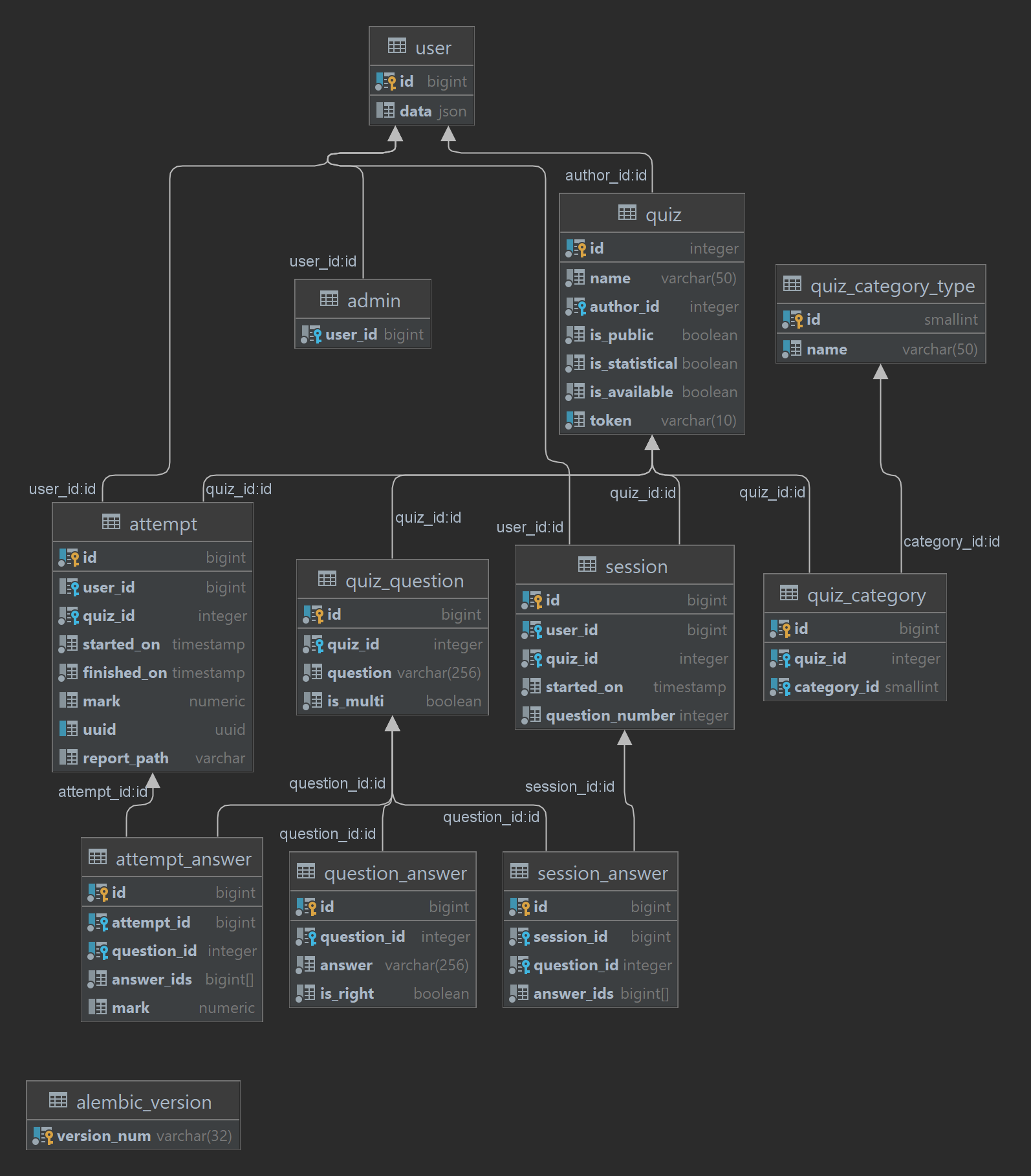


Рис. 2. 15. Діаграма бази даних.

## 2.3 Обробка даних системи

Обробка даних у системі утворена як гармонічне поєднання SQL-запитів та ORM-запитів, як і рекомендовано розробниками фреймворку, що лежить у основі взаємодії із базою даних у рамках цієї роботи – SQLAlchemy.

Першою із таких операцій, що буде розглянуто, є операція отримання та узагальнення даних щодо проходження тестування. А саме – статистика вірних, не 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 вирахувані 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 цій 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.

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

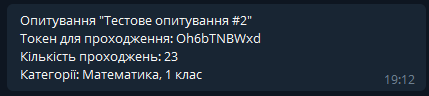
Створено операції, що аналізують дані та приводять результати цього аналізу у доступний користувачу вигляд. У результаті роботи цього коду користувач отримає графічно подану інформацію щодо своїх опитувань або спроби проходження тестування.

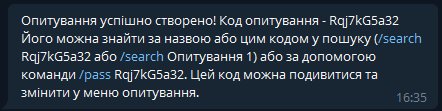
# РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПІДСИСТЕМИ ОБРОБКИ ДАНИХ СИСТЕМИ ОПИТУВАНЬ

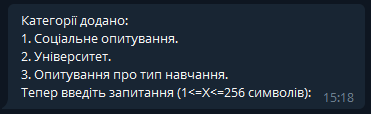
## 3.1 Проектування інтерфейсу обробки даних

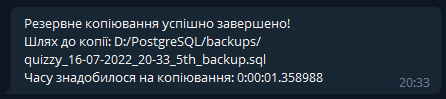
У обраній темі роботи проектування інтерфейсу у традиційному його визначенні не є можливим, оскільки у якості інтерфейсу використовується інтерфейс месенджера Telegram. У зв’язку із цим простір проектування звужується до трьох елементів: форматування тексту, прикріплення рисунків до повідомлень і клавіатура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

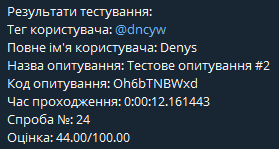
Перший елемент, форматування тексту, використано у всіх елементах взаємодії із користувачем. Далі наведено ті, з якими користувач найчастіше зустрічається:

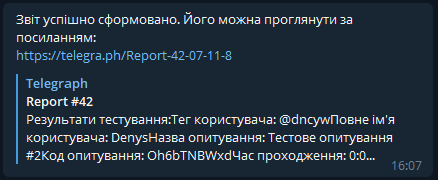
  
Рис. 3.1. Інформація про обране опитування.

  
Рис. 3.2. Інформація про новостворене опитування.

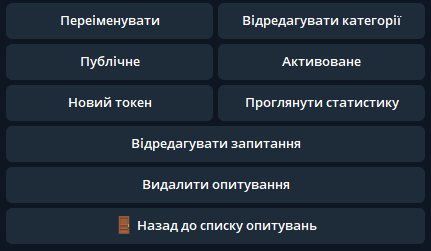
  
Рис. 3.3. Інформація про додані користувачем до опитування категорії.

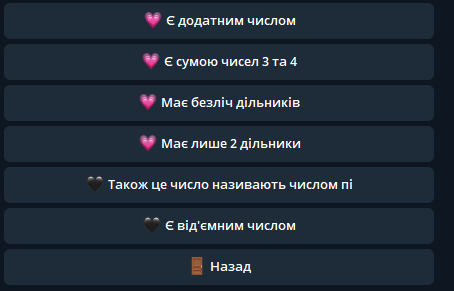
  
Рис. 3.4. Інформація про створену резервну копію.

  
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.

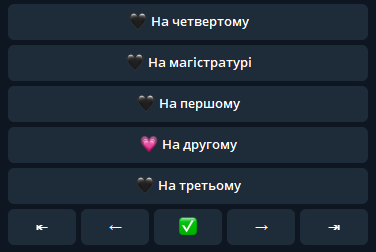
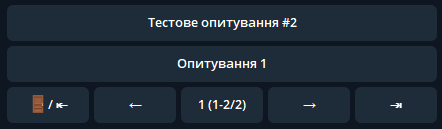
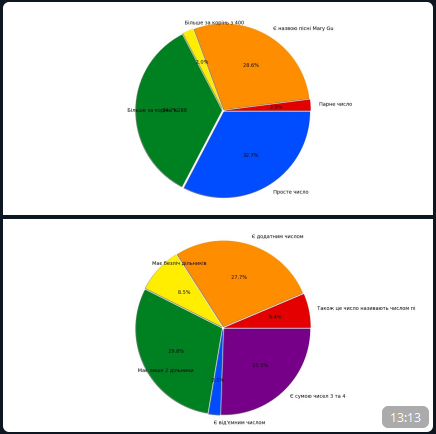


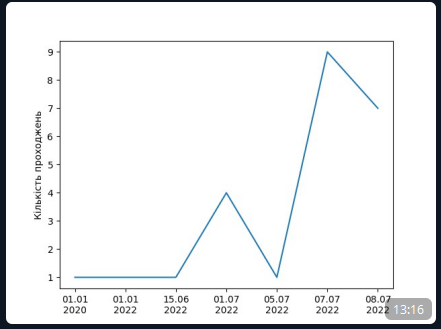
Рис. 3.9. Клавіатура проходження опитування.

  
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 буду 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')

1. 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')

1. 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)

1. 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')

1. 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')

1. 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)

1. 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)

1. 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)

1. 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)

1. 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')

1. Лістинг моделі AttemptAnswer:

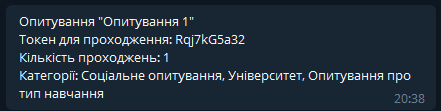
from sqlalchemy import Column, BigInteger, ForeignKey, Integer, UniqueConstraint, Numeric  
from sqlalchemy.orm import relationship  
from sqlalchemy.dialects.postgresql import ARRAY  
  
from db.models.Attempt import Attempt  
from db.models.QuestionAnswer import QuestionAnswer  
from db.models.QuizQuestion 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

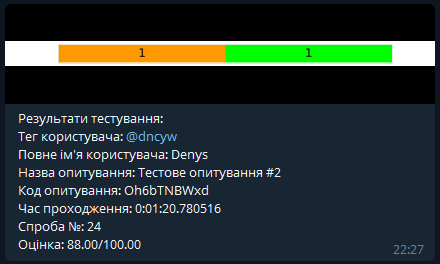
В проекті реалізовано декілька видів звітності, у тому числі ті, що вже частково були розглянуті в пунктах 2.3 та 3.1 та будуть розглянуті більш детально підчас розгляду організації звітності системи.

Серед інших у системі звітність реалізовано у наступних місцях:

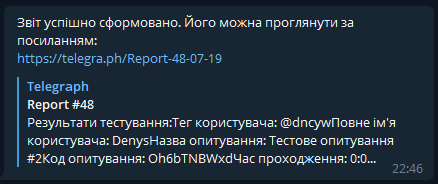
1. Відображення кількості проходжень, кількості людей, що прямо зараз проходять це опитування (відображається лише якщо такі є) та категорій цього опитування:

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

1. 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.
2. Відображення результатів проходження тестування у вигляді горизонтального стовпчикового 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.

1. Формування повного звіту по відповідям спроби проходження тестування:

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

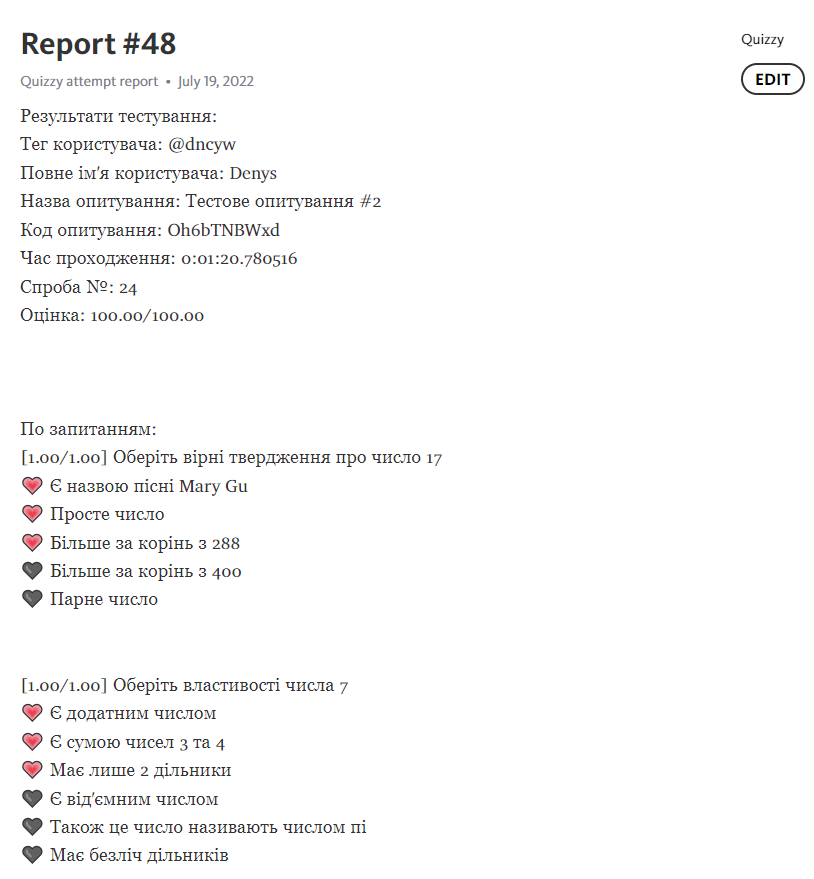
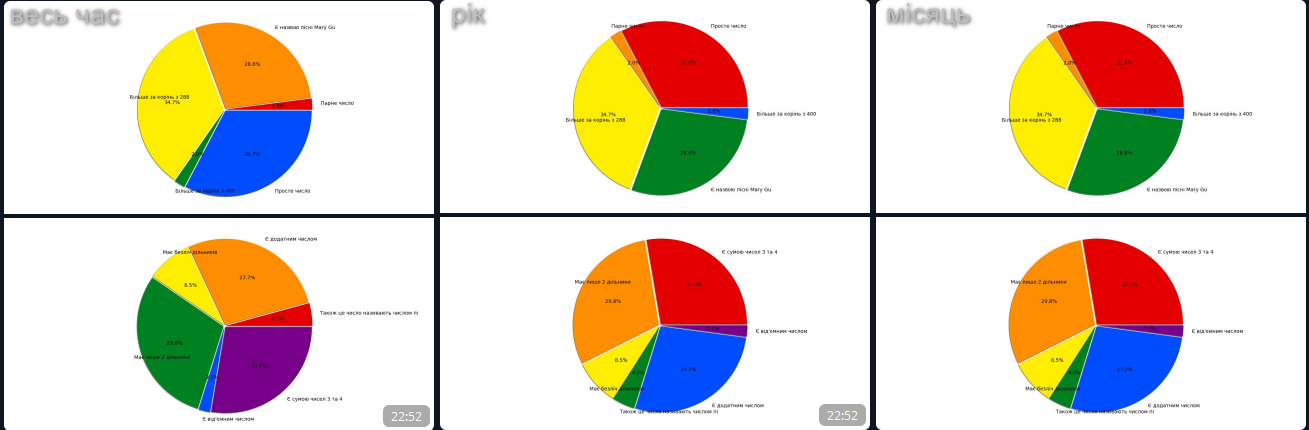
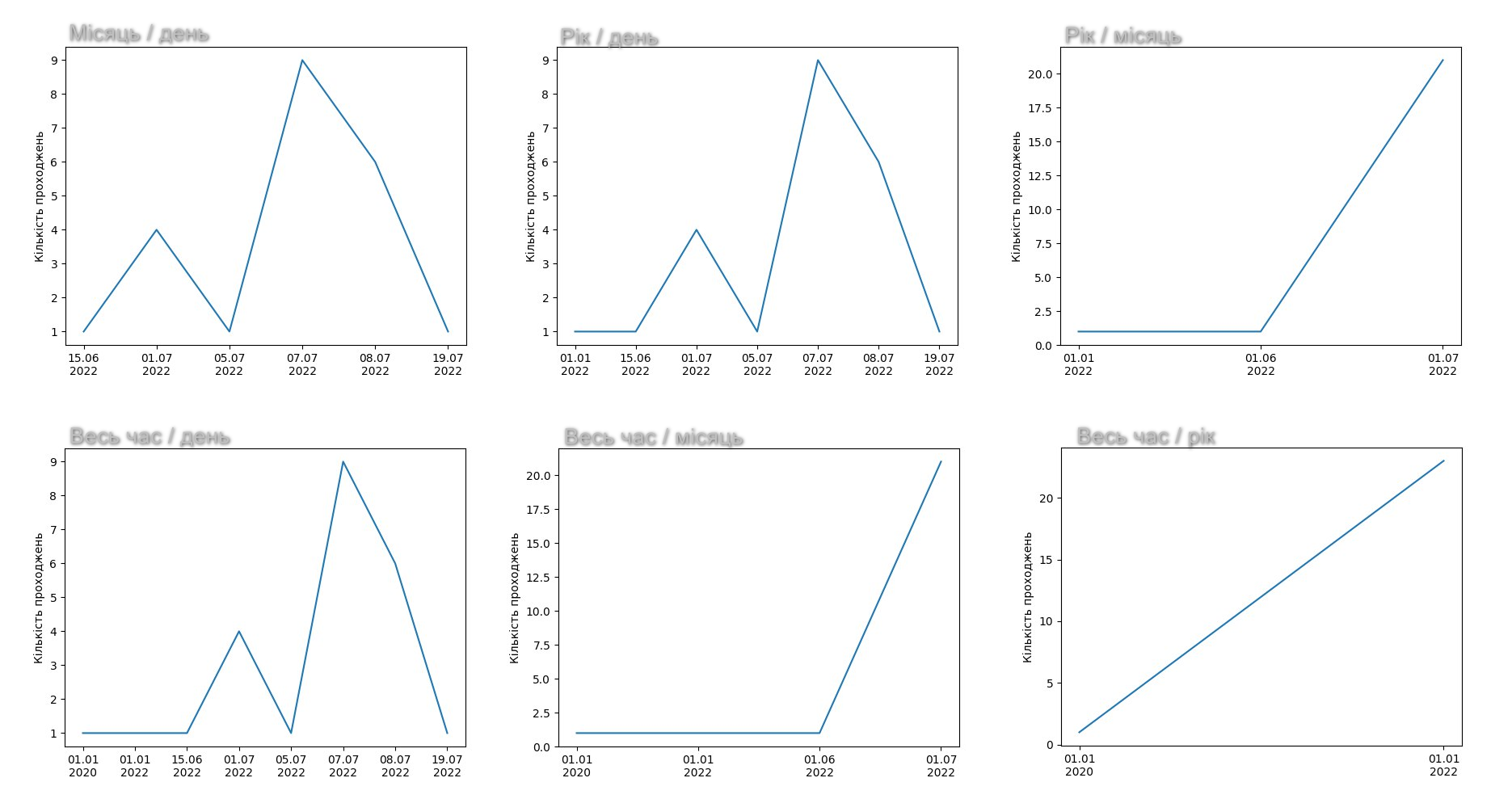


Рис. 3.17. Вигляд повного звіту, що знаходиться за посиланням на рис. 3.16.

1. 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.

1. 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:

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

## Висновки до 3 розділу:

Спроектовано 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:

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

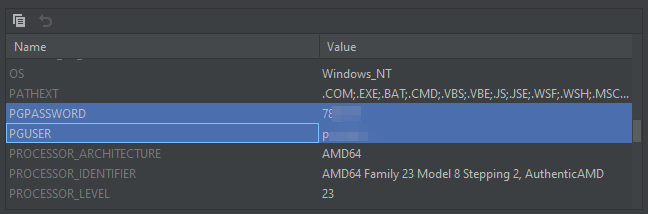
# replication privilege.

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

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

  
Рис. 4.1. Налаштування логіну та паролю за замовченням.

1. 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 між 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 підчас 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.

# ЛІТЕРАТУРА

1. Advanced Python Programming: Build high performance, concurrent, and multi-threaded apps with Python using proven design patterns – 320 c. – (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

## Додаток А

## 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.

Дисковій 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 підчас користування програмною системою та є її невід’ємною частиною. Керівництво та його відображення користувачу реалізовано підчас 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.

## Додаток Б

**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('Ні', 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 підчас 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(Attempt, attempt\_id)  
 quiz = s.get(Quiz, attempt.quiz\_id)  
 if not quiz.is\_statistical:  
 retry\_number = s.query(Attempt).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(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)  
  
 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 nagative)  
 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 as 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))