

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	2
1. Анализ предметной области	3
1.1 Первичная постановка задачи	3
1.2 Анализ предметной области	3
1.3 Анализ возможных методов решения	3
1.4 Цели и задачи разработки	4
программы	4
2. Разработка требований к программе	5
2.1 Требования к пользовательскому интерфейсу	5
2.2 Требования к программному интерфейсу и используемым ресурсам	8
3. Разработка программы	9
3.1 Проектирование структуры программы	9
3.2 Описание данных и структур данных	9
3.2.1 Описание базы данных	10
3.2.2 Данные, запрашиваемые у пользователя	12
3.3 Разработка алгоритмов	13
3.4 Реализация и отладка программы	15
4. Тестирование	23
ЗАКЛЮЧЕНИЕ	34
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	35
ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММНОГО КОДА	36

ВВЕДЕНИЕ

Мы живём в век информационных технологий и прогресса в этой сфере, одно множество людей из-за такого информационного развития живёт довольно малоактивной жизнью и как следствие совершенно забывают о базовых занятиях физической активностью, а другое множество может заниматься спортом, но и делает это неэффективно, например, не имея системного плана своей активности.

Продукт данной работы призван помочь в этой ситуации и составлять план тренировок на основе предпочтений пользователя и находить нужные планы под нужды, а также делиться с остальными пользователями своими активностями, позволяя внедрять здоровый образ жизни в массы.

Целью курсовой работы по дисциплине “Основы программирования” является создание инструмента, в данном случае Telegram-бота, который позволяет формировать план тренировок под нужды пользователя и делиться им со всеми пользователями.

Для выполнения цели будет использоваться средство разработки язык программирования Python 3 и инструмент хранения данных на сервере в виде реляционной базы данных MySQL.

1. Анализ предметной области

1.1 Первичная постановка задачи

Задача на курсовую работу: Разработка Telegram-бота “План индивидуальных тренировок” без номера варианта.

1.2 Анализ предметной области

Обрабатываемые данные: Виды тренировок, выбранные пользователем. Данные возраста, роста и веса, введенные пользователем. Общая база видов тренировок, предлагаемая по умолчанию.

Цель обработки: сформировать цельный и полноценный план под требования пользователя.

1.3 Анализ возможных методов решения

В качестве возможных методов решения будут рассматривать методы хранения информации:

- MySQL сервер
- Файловое хранение на сервере
- Файловое хранение на клиенте

Для выбора подходящего способа хранения будет сравниваться краткая характеристика данных способов.

- Файловое хранение на клиенте позволяет хранить информацию необходимую для обработки на стороне пользователя. Способ является не безопасным, так как пользователь может в любой момент отредактировать файл хранения, повредив его или удалив, что приведёт к потере данных.
- Файловое хранение на сервере позволяет хранить все файлы с данными о пользователях на сервере. Программа сможет находить файл по уникальному идентификатору и выгружать информацию

пользователю. Этот метод надёжный с точки зрения безопасности хранения данных и безопасности.

- MySQL позволяет хранить всю информацию в базе данных MySQL, где каждый пользователь будет храниться под уникальным идентификатором. Метод такой же надёжный, как и файлы, но выигрывает в оптимизации запросов на поиск и изменение данных.

Среди трёх методов хранения данных решено выбрать MySQL за его быстродействие и удобство в работе с информацией.

1.4 Цели и задачи разработки программы

Цель разработки программы: Предоставление помощи в составлении плана индивидуальных тренировок пользователя

Задачи разработки программы:

- Составление профиля пользователя.
- Создание плана тренировок из предложенных ботом или пользовательских.
- Создание личной заметки.
- Загрузка/Удаление плана тренировки в/из общую базу.
- Выгрузка чужого плана в свой план.
- Написание сообщений для разработчика через модуль обратной связи.

2. Разработка требований к программе

2.1 Требования к пользовательскому интерфейсу

В программе реализован метод общения с пользователем посредством выбора элементов меню и ввода информации.

При запуске Telegram-бота с помощью команды /start программа начинает диалог с пользователем и возможно 3 исхода:

1. Если какие-то данные (рост, вес, возраст) об пользователе отсутствуют в разделе Профиль в базе данных.

Вывод меню:

1. Рост
2. Вес
3. Возраст

При выборе каждого из пунктов выполняется запрос у пользователя выбранного данного. На ввод существуют ограничения в соответствии с разделом 3.2 Описание данных и структур данных.

После ввода всех данных пользователю предлагается перейти в главное меню (подробнее в исходе 2).

2. Если данные в разделе Профиль в базе данных присутствуют.

Вывод меню:

1. План тренировок
2. База готовых планов
3. Профиль
4. Об авторе

При выборе пункта меню №1:

Вывод списка тренировок пользователя и личной заметки.

1. Меню:

1. Добавить пункт плана
2. Редактировать заметку
3. Назад

1.1 Пункт №1 доступен, только если количество пунктов меньше 15. При выборе предлагается написать тренировку вручную, либо выбрать из базы, а также вернуться в Главное меню.

1.2 При выборе пункта №2 предлагается ввести текст личной заметки. Ограничение на текст в соответствии с разделом 3.2 Описание данных и структур данных.

1.3 Пункт №3 возвращает в Главное меню.

При выборе пункта меню №2:

Вывод списка планов тренировок в базе готовых планов, 10 экземпляров на страницу.

2. Меню:

1. Назад
2. Вперёд
3. Мои планы
4. Просмотреть пункт
5. В меню

Пункты 1 и 2 выполняют функцию пролистывания списка планов и доступны, только если имеется возможность пройти назад/вперёд по списку.

2.3 При выборе пункта №3 выводится список планов, опубликованных пользователем.

2.3 Меню:

1. Загрузить текущий план
2. Удалить план
3. Просмотреть пункт
4. Назад

2.3.1 Пункт №1 доступен, если планов меньше 10 и программа запрашивает название и описание плана, проверяя уникальность имени

в планах данного пользователя. В случае успеха выкладывает план, иначе просит повторить ввод. Доступна возможность выхода в главное меню.

2.3.2 Пункт №2 доступен, если существует хотя бы один план. При выборе пункта запрашивает порядковый номер плана в списке. В случае нахождения удаляет его из базы, иначе запрашивает повторный ввод. Доступна возможность выхода в главное меню.

2.3.3 Пункт №3 запрашивает порядковый номер плана в списке. В случае нахождения плана показывает информацию об плане (Имя, описание, состав плана, создателя). Доступны действия загрузки в текущий план пользователя, удаления из базы, а также возвращения в Главное меню. В случае не нахождения плана по номеру запрашивает номер повторно и доступен выход в Главное меню.

2.3.4 Пункт №4 возвращает в Главное меню.

2.4 При выборе пункта №4 позволяет просмотреть план, как описано в пункте 2.3.3, но пункт Удалить план доступен, только если пользователь является автором.

2.5 При выборе пункта №5 возвращает в Главное меню.

При выборе пункта меню №3:

Вывод информации о пользователе (Аватар, рост, вес, возраст)

3. Меню:

1. Редактировать
2. Назад

3.1 Пункт меню №1 пользователю предлагается изменить один из пунктов профиля (Рост, вес, возраст), а также вернуться в Главное меню

3.2 Пункт меню №2 пользователь возвращается в Главное меню.

При выборе пункта меню №4:

Вывод информации об авторе.

4. Меню:

1. Обратная связь
2. Назад

4.1 Пункт меню №1 пользователю предоставляется возможность написать сообщение автору или же вернуться в Главное меню.

4.2 Пункт меню №2 возвращает в Главное меню.

2.2 Требования к программному интерфейсу и используемым ресурсам

Для использования данной программы-бота пользователю необходимо иметь аккаунт в мессенджере Telegram, а также возможность набора текста и нажатия на варианты ответа любым поддерживаемым Telegram способом.

Сервер, на котором стоит база данных, которая используется программой, должен быть активным в течении всего времени, иначе программа не будет иметь доступа к данным пользователей до момента повторной активации сервера. Так же сервер должен иметь достаточную пропускную способность для обработки нескольких запросов от одновременных запросов разных пользователей.

3. Разработка программы

3.1 Проектирование структуры программы

Программа для курсовой работы по дисциплине “Основы программирования” пишется на языке Python версии 3 и использованием библиотеки Python Telegram Bot версии 21.7, которая как раз позволяет создать Telegram-бота на языке программирования Python, взаимодействуя с Telegram API. Так же был написан собственный модуль под названием `tg_sql`, содержащий функции работы с базой данных, которая необходима для работы программы, он был написан на основе модуля работы с базой данных MySQL под названием MySQL Connector Python. Графическое представление взаимодействия программы с модулями и пользователем представлено на рисунке 1.

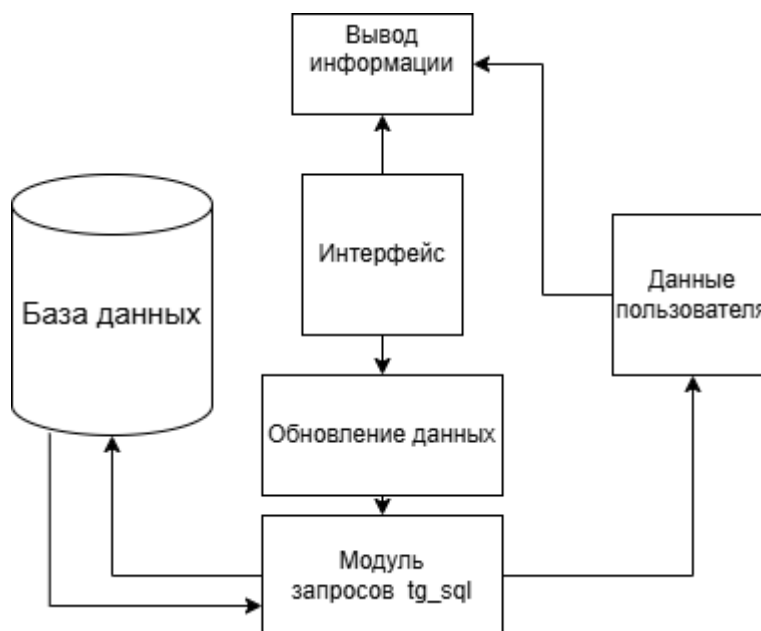


Рисунок 1 – Графическое представление программы

3.2 Описание данных и структур данных

В данной курсовой работе используется реляционная база данных на основе MySQL. Далее данная база данных и её использование будут описаны.

3.2.1 Описание базы данных

База данных имеет следующие таблицы хранения данных, графическое представление показано на рисунке 2:

- Данные пользователей users
- База тренировок trains
- Общая база тренировок пользователей trains_users

trains_users ссылается внешним ключом на таблицу пользователей users, которая является главной.

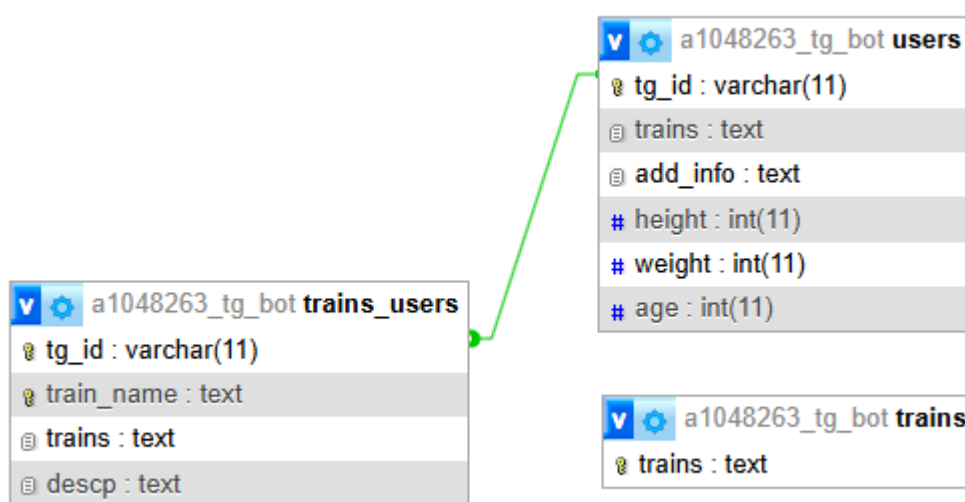


Рисунок 2 – База данных

Таблица данных users хранит данные каждого пользователя, который использовал программу, информация о полях представлена в таблице 1.

Таблица 1 - users

Имя поля	Тип данных	Семантика
tg_id	string	уникальный идентификатор (первичный ключ) пользователя
trains	string	Массив с тренировками в виде JSON

		строки. Каждая тренировка не более 150 символов
add_info	string	Дополнительная информация пользователя
height	int	Рост указанный пользователем
weight	int	Вес указанный пользователем
age	int	Возраст указанный пользователем

Таблица данных trains является местом хранения базы тренировок, которые предлагаются пользователю при заполнении его плана. Информация о полях представлена в таблице 2.

Таблица 2 - trains

Имя поля	Тип данных	Семантика
trains	string	Название тренировки

Данная таблица является общей базой данных всех тренировок на сервере, из которых пользователь может найти по своему вкусу и добавить в себе в план.

Модификация таблицы производится только разработчиком.

Таблица данных trains_users является местом хранения тренировок, которые пользователи выкладывают в общий доступ. Информация о полях представлена в таблице 3.

Таблица 3 - trains_users

Имя поля	Тип данных	Семантика
tg_id	string	Уникальный идентификатор пользователя
train_name	string	Название плана тренировки

trains	string	Массив с тренировками в виде JSON строки
descp	string	Описание плана тренировки

Уникальность данных проверяется в комбинации tg_id + train_name.

3.2.2 Данные, запрашиваемые у пользователя

Данные, которые запрашиваются у пользователя представлены в таблице 4.

Таблица 4 – Запрашиваемые данные.

Наименование	Тип данных	Семантика	Описание проводимых проверок корректности данных
age	int	Возраст пользователя	Целое число в диапазоне 0 - 120
height	int	Рост пользователя	Целое число в диапазоне 70 - 255
weight	int	Вес пользователя	Целое число в диапазоне 20 - 500
note	string	Личная заметка	Текст длиной <= 150
train_name	string	Название тренировки для добавления	Текст длиной <= 150
train_descp	string	Описание выкладываемого плана	Текст длиной <= 150
num	int	Порядковый номер в списке тренировок	Целое число в диапазоне 1 – максимальный номер в списке

3.3 Разработка алгоритмов

По большей своей части программа состоит из запросов к базе данных MySQL, и присутствуют лишь небольшие вспомогательные алгоритмы, которые будут представлены ниже.

Блок-схема функции поиска тренировки по порядковому номеру в списке представлена на рисунке 3.

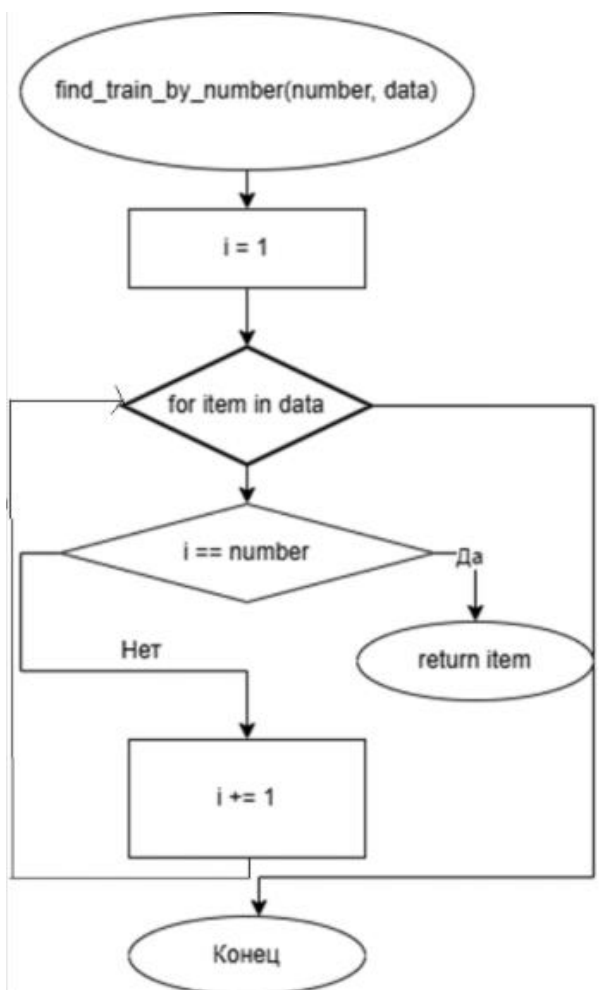


Рисунок 3 – функция `find_train_number`

Блок-схема функции поиска среди тренировок пользователя по имени представлена на рисунке 4.

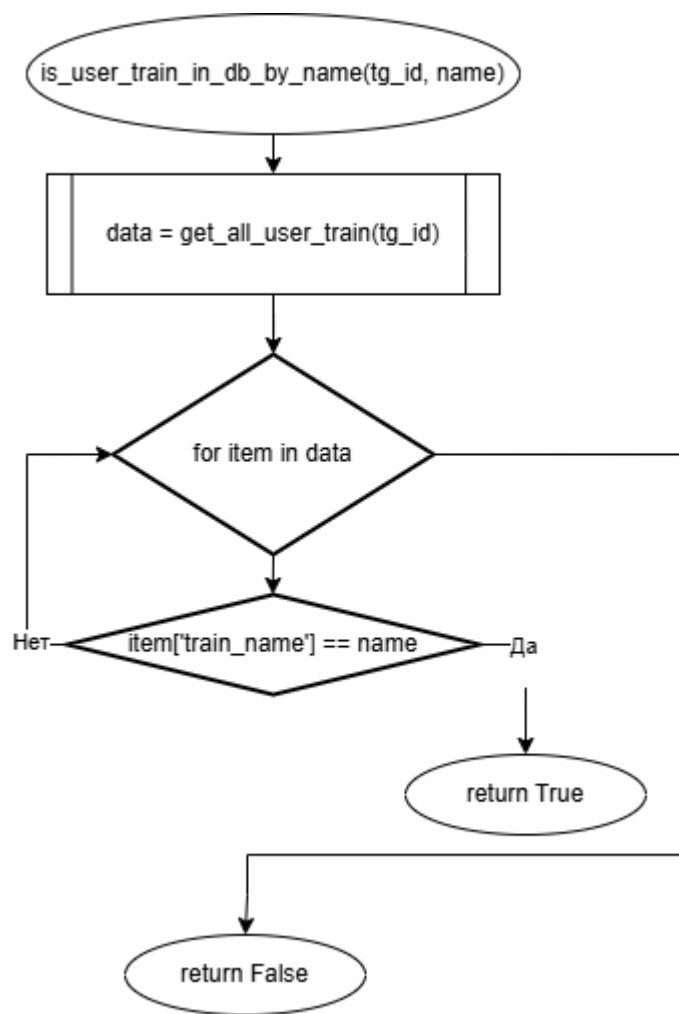


Рисунок 4 – `is_user_train_in_db_by_name`

Блок-схема функции проверки, есть ли в базе хоть какая-то информация о пользователе представлена на рисунке 5.

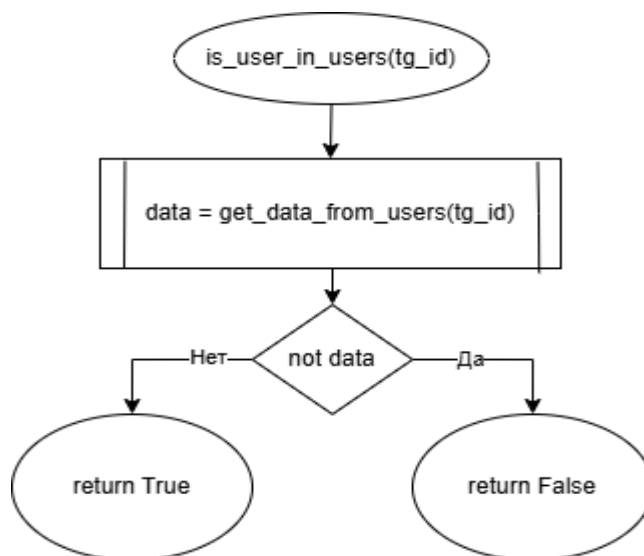


Рисунок 5 – `is_user_in_users`

3.4 Реализация и отладка программы

В данной курсовой работе используется конвенция именования переменных, функций и модулей под названием PEP8, которая разработана специально для языка программирования Python.

Весь программный код был написан с использованием программы PyCharm Community Edition и его встроенными методами отладки кода.

В процессе работы был специально разработан модуль `tg_sql`, который содержит все функции взаимодействия программы с базой данных, на основе модуля работы с базой данных MySQL под названием MySQL Connector Python версии 9.1.0.

Далее будут описаны все функции, которые были разработаны в процессе написания программы. Описание начнётся с модуля `tg_sql` и представлено в таблице 5.

Таблица 5 – Перечень разработанных функций в модуле `tg_sql`

№	Прототип	Входные параметры	Выходные параметры	Описание функционала
1	<code>def check_server</code>	-	bool True – подключено False – нет подключения	Проверка сервера на подключение к базе данных
2	<code>def check_and_retry_connection</code>	-	-	Проверка соединения с базой данных и в случае неудачи попытается

				присоединиться
3	def is_user_train_in_bd_by_name(tg_id, name)	name – string – имя плана	bool True – тренировка существует False – тренировки не существует	Проверка, есть ли в базе планов пользователя такое имя
4	def is_user_in_users(tg_id)	tg_id – string – Telegram ID	bool True – Данные о пользователе есть False – Данных нет	Проверка, есть ли пользователь в базе
5	def is_user_profile_contain_trains(tg_id)	tg_id – string – Telegram ID	bool True – Данные о тренировках есть False – Данных нет	Проверка, есть ли у пользователя тренировки
6	def is_user_profile_full(tg_id)	tg_id – string – Telegram ID	bool True – Данные полные False –	Проверка, полные ли данные (возраст, рост, вес) профиля у

			Данные неполные	пользователя
7	def get_data_from_users(tg_id)	tg_id – string – Telegram ID	Python Dict – информация пользователя	Получение всех данных пользователя из users
8	def set_user_age(tg_id, age)	tg_id – string – Telegram ID age - int – Возраст	-	Установка возраста
9	def set_user_weight(tg_id, weight)	tg_id – Telegram ID weight - int – Вес	-	Установка веса
10	def set_user_height(tg_id, height)	tg_id – Telegram ID height - int – Рост	-	Установка роста
11	def set_user_info(tg_id, add_info)	tg_id – Telegram ID add_info – string – Личная заметка	-	Установка личной заметки
12	def set_user_trains(tg_id, trains)	tg_id – string – Telegram ID trains – string	-	Установка тренировок

		– Тренировки		
13	def get_all_users_trains_by_page	-	Python Dict – Планы тренировок	Получение всех выложенных планов
14	def get_all_user_train(tg_id)	tg_id – string – Telegram ID	Python Dict – Планы тренировок	Получение всех выложенных планов пользователя
15	def upload_user_train(tg_id, name, trains, desc)	tg_id – string – Telegram ID name – string – Имя плана trains – string – Список тренировок descp – string – Описание плана	-	Загрузка пользовательского плана в базу данных
16	def delete_user_train_by_name(tg_id, name)	tg_id – string – Telegram ID name – string – Имя плана	-	Удаление пользовательского плана
17	def get_user_train_by_name(tg_id, name)	tg_id – string – Telegram ID name – string	Python Dict – План тренировок	Получение конкретного пользовательского

		– Имя плана		плана
18	def get_all_trains()	-	-	Получение все тренировок из общей базы тренировок

Описание разработанных функций в основном файле программы main представлено в таблице 6.

Из-за повторения входных данных в связи с особенностью библиотеки Python Telegram Bot, следующие входные не будут повторно описываться:

update – Update – Объект содержащий входящее сообщение

context – CallBack Context – Объект, содержащий информацию обратного вызова

Таблица 6 – Функции основного файла

№	Прототип	Входные параметры	Выходные параметры	Описание функционала
1	find_train_by_number(number, data)	number – int – порядковый номер data – Python Dict – Список планов/тренировок	Python Dict – информация о тренировке/плане	Возвращает план/тренировку по порядковому номеру в списке
2	def setup_user_data(update, context)	update, context	-	Установка информации о пользователе в

				память
3	async def start(update, context):	update, context	int – Статус общения	Функция начала общения
4	async def menu_of_data(update, context, text="Выберите пункт редактирования")	update, context, text – string – Текст для вывода в меню	int – Статус общения	Меню заполнения Рост-Вес-Возраст
5	async def editing(update, context)	update, context	int – Статус общения	Меню ввода редактируемой информации
6	async def save_info(update, context)	update, context	int – Статус общения	Проверка и сохранение вводимых данных
7	def saving(update, context)	update, context	-	Сохранение данных в базе
8	async def menu(update, context)	update, context	int – Статус общения	Главное меню
9	async def author(update, context)	update, context	int – Статус общения	Меню Об авторе
10	async def feedback(update, context)	update, context	int – Статус общения	Меню написания сообщения через обратную связь
11	async def send_feed(update,	update,	int – Статус	Функция

	context)	context	общения	отправления сообщения по обратной связи
12	async def profile(update, context)	update, context	int – Статус общения	Меню профиль пользователя
13	async def base_train(update, context)	update, context	int – Статус общения	Меню база планов, выложенных пользователями
14	async def self_trains_menu(update, context)	update, context	int – Статус общения	Список опубликованных планов пользователя
15	async def show_exact_trains(update, context)	update, context	int – Статус общения	Вывод конкретного плана
16	async def delete_exact_train(update, context)	update, context	int – Статус общения	Удаление конкретного плана
17	async def load_exact_train(update, context)	update, context	int – Статус общения	Загрузка плана вместо пользовательского
18	async def show_plan(update, context)	update, context	int – Статус общения	Меню пользовательского плана
19	async def upload_trains(update,	update,	int – Статус	Вспомогательная функция загрузки

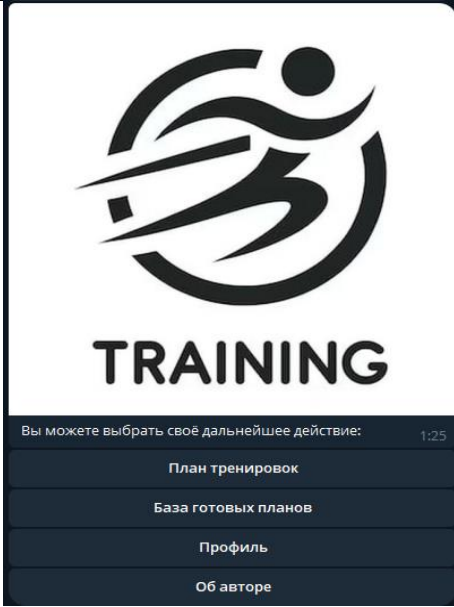
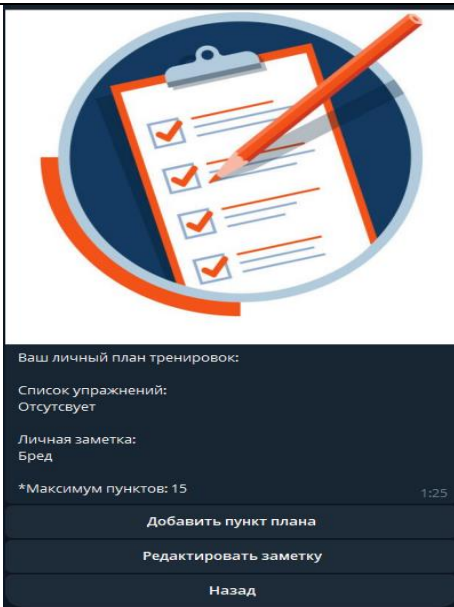
	context)	context	общения	плана в базу
20	async def stop(update, context)	update, context	int – Статус общения	Остановка общения в случае непредвидимой ошибки
21	async def inline_query_res(update, context)	update, context	-	Предложение тренировки программой посредством telegram inline query
22	def main()	-	-	Запуск и инициализация команд


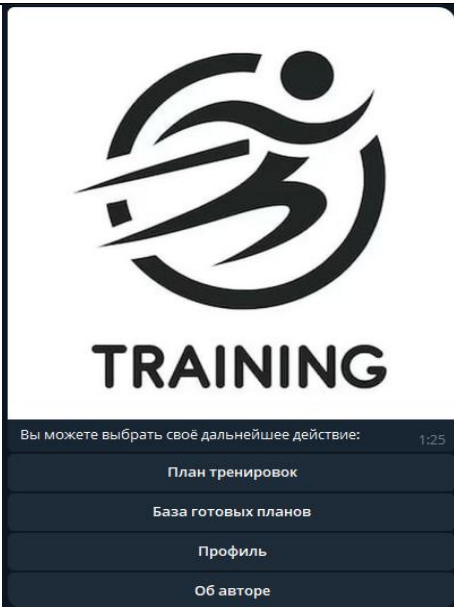
Полный код программы, написанный в процессе курсовой работы, можно найти в Приложении А Листинг прораммного кода.


4. Тестирование



Для проверки корректности выполнения работы программы, написанной для курсовой работы, были проведены тесты и использованы тестовые данные. Все сценарии тестирования указаны в таблице 7.




Таблица 7 – Тестирование программы.

№	Название этапа тестирования	Тестовые данные	Ожидаемый результат	Фактический результат
1	Главное меню программы. Ввод команды /start	-	Вывод главного меню	
2	План тренировок. Выбор 1 кнопки	-	Вывод плана тренировок и действий с ним	

3	Добавление пункта плана.	-	Запрос текста для пункта	 <p>Введите текст пункта:</p> <p>*Вы можете предложить добавить в базу тренировку *Об авторе -> Обратная связь 1:25</p> <p>Назад</p> <p>Выбрать из базы ↗</p>
4	Возвращение в главное меню. Кнопка Назад	-	Меню Главное меню	 <p>Вы можете выбрать своё дальнейшее действие: 1:25</p> <p>План тренировок</p> <p>База готовых планов</p> <p>Профиль</p> <p>Об авторе</p>
5	Ввод текста более 150 символов	Строка из букв 'а' длиной 151	Ошибка и просьба ввести заново	 <p>Текст слишком длинный! Повторите попытку: 1:42</p>

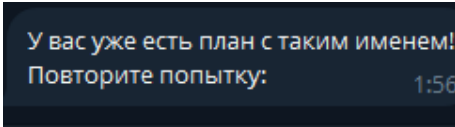
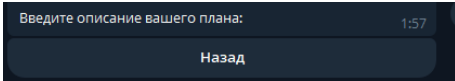
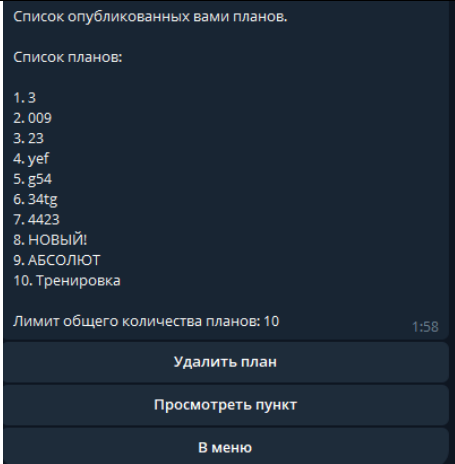
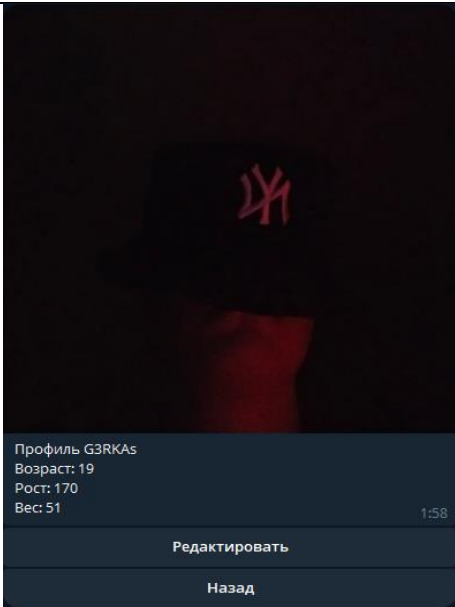
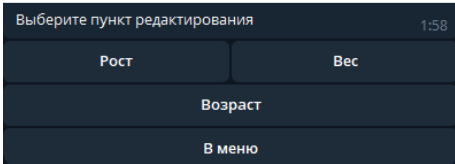
6	Ввод верного текста	Подтягивания	Добавление в план и вывод плана	 <p>Ваш личный план тренировок:</p> <p>Список упражнений: 1. Подтягивания</p> <p>Личная заметка: Бред</p> <p>*Максимум пунктов: 15 1:43</p> <p>Добавить пункт плана Удалить пункт плана</p> <p>Редактировать заметку</p> <p>Назад</p>
7	Выбор тренировки из базы	-	Вывод списка тренировок	<p>Тяга верхнего блока</p> <p>Отжимания на брусьях</p> <p>Разгибания рук на трицепс</p> <p>Жим лежа</p> <p>Становая тяга</p> <p>Приседания со штангой</p> <p>Подтягивания</p> <p>Тяга штанги в наклоне</p> <p>Армейский жим</p> <p>Выпады</p> <p>Отжимания</p> <p>Меню @plan_train_bot Название тренировки</p>

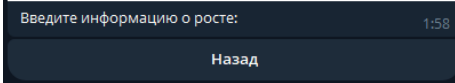
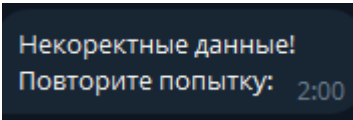
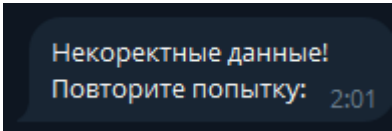
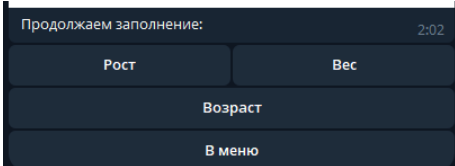
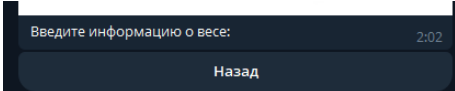
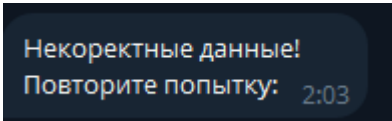
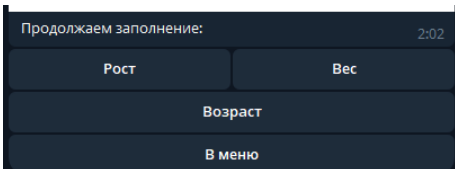
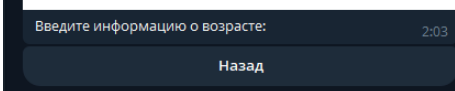
8	Ввод выбранной тренировки	@plain_train_ bot Приседания со штангой	Добавление в план и вывод плана	 <p>Ваш личный план тренировок:</p> <p>Список упражнений: 1. Подтягивания 2. Приседания со штангой</p> <p>Личная заметка: Бред</p> <p>*Максимум пунктов: 15 1:45</p> <p>Добавить пункт плана Удалить пункт плана</p> <p>Редактировать заметку</p> <p>Назад</p>
9	Кнопка Удалить пункт плана	-	Вывод просьбы ввести пункт для удаления	 <p>TRAINING</p> <p>Введите номер пункта:</p> <p>1. Подтягивания 2. Приседания со штангой 1:45</p> <p>Назад</p>
10	Ввод неверного пункта	3	Вывод об ошибке и просьба ввести заново	<p>Такого пункта не существует! Повторите попытку: 1:48</p>

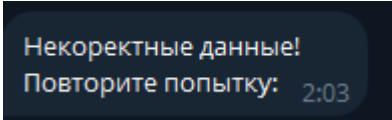
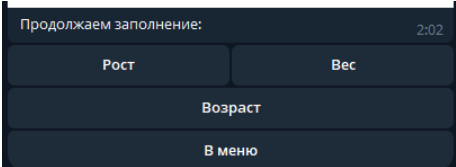
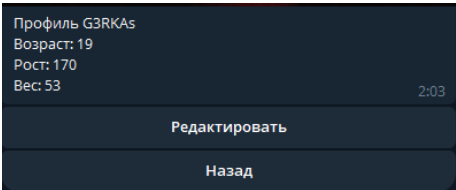
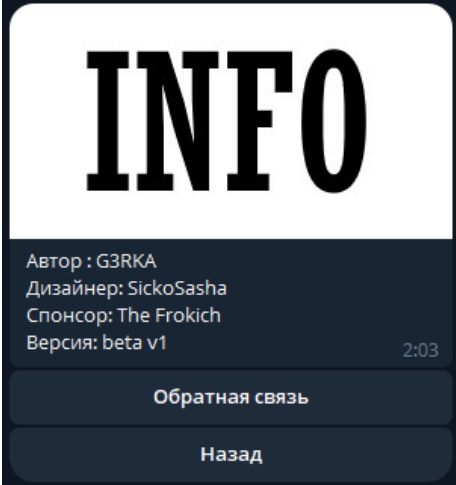
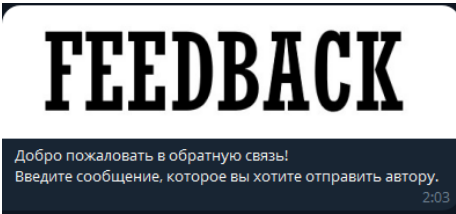
11	Ввод верного пункта	2	Удаление тренировки и вывод плана	 <p>Ваш личный план тренировок:</p> <p>Список упражнений: 1. Подтягивания</p> <p>Личная заметка: Бред</p> <p>*Максимум пунктов: 15 1:48</p> <p>Добавить пункт плана Удалить пункт плана</p> <p>Редактировать заметку</p> <p>Назад</p>
12	Кнопка Редактировать заметку	-	Вывод просьбы ввести текст для заметки	 <p>TRAINING</p> <p>Введите желаемую заметку: 1:48</p> <p>Назад</p>
13	Ввод правильной заметки	Вставить утром в 9:00	Добавление заметки и вывод плана	 <p>Ваш личный план тренировок:</p> <p>Список упражнений: 1. Подтягивания</p> <p>Личная заметка: Вставать утром в 9:00</p> <p>*Максимум пунктов: 15 1:50</p> <p>Добавить пункт плана Удалить пункт плана</p> <p>Редактировать заметку</p> <p>Назад</p>

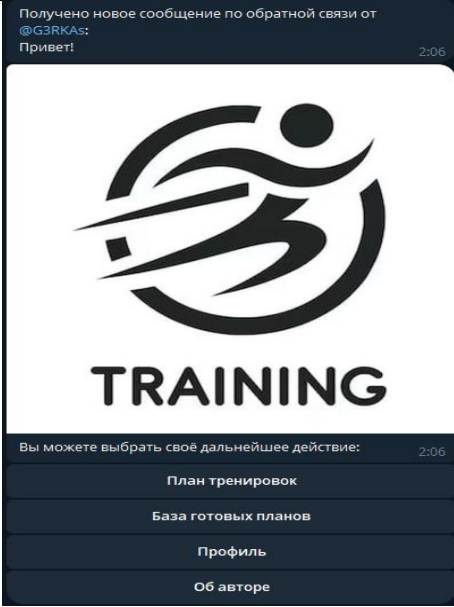
14	Кнопка База ГОТОВЫХ ПЛАНОВ	-	Меню с базой ГОТОВЫХ ПЛАНОВ	<p>Открытая база планов тренировок от других пользователей.</p> <p>Список планов:</p> <ol style="list-style-type: none"> 1. PK от Ducker 2. ;"№KAA от Ducker 3. 43 от Ducker 4. 3 от G3rka 5. 009 от G3rka 6. 23 от G3rka 7. yef от G3rka 8. g54 от G3rka 9. 34tg от G3rka 10. 4423 от G3rka <p>1:50</p> <p>Вперёд</p> <p>Мои планы</p> <p>Просмотреть пункт</p> <p>В меню</p>
15	Кнопка Вперёд	-	Вывод следующей страницы	<p>Открытая база планов тренировок от других пользователей.</p> <p>Список планов:</p> <ol style="list-style-type: none"> 1. НОВЫЙ! от G3rka 2. АБСОЛЮТ от G3rka 3. XAI от Ducker <p>1:50</p> <p>Назад</p> <p>Мои планы</p> <p>Просмотреть пункт</p> <p>В меню</p>
16	Кнопка Назад	-	Вывод предыдущей страницы	<p>Открытая база планов тренировок от других пользователей.</p> <p>Список планов:</p> <ol style="list-style-type: none"> 1. PK от Ducker 2. ;"№KAA от Ducker 3. 43 от Ducker 4. 3 от G3rka 5. 009 от G3rka 6. 23 от G3rka 7. yef от G3rka 8. g54 от G3rka 9. 34tg от G3rka 10. 4423 от G3rka <p>1:50</p> <p>Вперёд</p> <p>Мои планы</p> <p>Просмотреть пункт</p> <p>В меню</p>

17	Кнопка Просмотреть пункт	-	Просьба ввести номер плана	<div> Введите номер плана: <div> 1. PK от Ducker 2. ;"№KAA от Ducker 3. 43 от Ducker 4. 3 от G3rka 5. 009 от G3rka 6. 23 от G3rka 7. yef от G3rka 8. g54 от G3rka 9. 34tg от G3rka 10. 4423 от G3rka </div> <div>Назад</div> </div>
----	--------------------------------	---	-------------------------------	---

22	Ввод существующего имени	4423	Вывод ошибки и просьба ввести заново	
23	Ввод корректного имени	Тренировка	Просьба ввести описание плана	
24	Ввод корректного описания	Хорошая тренировка	Добавление плана в базу и вывод опубликованных	
25	Кнопка Профиль	-	Вывод профиля пользователя	
26	Кнопка Редактировать	-	Вывод возможных данных для редактирования	

27	Кнопка Рост	-	Просьба ввести рост	
28	Ввод некорректной информации	пп	Вывод ошибки и просьба ввести заново	
29	Ввод информации вне ограничения	999	Вывод ошибки и просьба ввести заново	
30	Корректный ввод	170	Обновление данных и вывод меню заполнения	
31	Кнопка Вес	-	Просьба ввести вес	
32	Ввод информации вне ограничения	0	Вывод ошибки и просьба ввести заново	
33	Ввод корректной информации	53	Обновление данных и вывод меню заполнения	
34	Кнопка Возраст	-	Просьба ввести возраст	

35	Ввод информации вне ограничения	190	Вывод ошибки и просьба ввести заново	
36	Ввод корректной информации	19	Обновление данных и вывод меню заполнения	
37	Возвращение в профиль. Кнопка Назад	-	Показ обновлённой информации в профиле	
38	Кнопка Об авторе.	-	Вывод информации об авторе	
39	Кнопка Обратная связь	-	Предложение написать сообщение в обратную связь	

40	Отправление сообщения по обратной связи	Привет!	Отправление сообщения и возвращение в главное меню	
----	---	---------	--	---

Протестировав всю программу на отклонения и не обнаружив ошибок и багов, был сделан вывод, что программа написана успешно и в точности пункта 2.1 Требования к пользовательскому интерфейсу.

ЗАКЛЮЧЕНИЕ

Результатом курсовой работы по дисциплине “Основы программирования” стал Telegram-бот, доступный по ID @plan_train_bot и связанная с ним реляционная база данных. Он предназначен для создания пользовательского плана тренировки и выкладывание его в общую базы для остальных пользователей.

В данном Telegram-боте имеется возможность управления своим профилем, а именно установка роста, возраста и веса, возможность обратной связи с разработчиком, возможность создания плана тренировки на основе собственных тренировок или предложенных пользователем и личной заметки, возможность просмотра общей базы данных планов и загрузка/выгрузка планов.

Разработанный инструмент в виде Telegram-бота позволяет без проблем составлять собственный план тренировок и изменения его в будущем. В перспективе обновлений бота находятся данные пункты:

- Обновление общей базы тренировок
- Добавление нового функционала
- Исправление возможных багов и ошибок работы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Туманова А.В. Фоменкова А.А. Ключарёв А.А. Основы программирования. Основные этапы разработки программ. Учебное пособие. СПб. ГУАП. 2014 год. Дата обращения: 19.11.2024
2. devs.mysql.com MySQL Develover Zone. Oracle. 2024 год. Дата обращения: 19.11.2024
3. docs.python-telegram-bot.org Python Telegram Bot Wiki. Leandro Toledo. 2024. Дата обращения: 19.11.2024
4. python.org Python. The Python Software Foundation. 2024 год. Дата обращения: 19.11.2024
5. telegram.org Telegram Messenger. Telegram FZ LLC. 2024 год. Дата обращения: 19.11.2024
6. core.telegram.org Telegram APIs. Telegram FZ LLC. 2024 год. Дата обращения: 19.11.2024

ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММНОГО КОДА

Файл main.py

```
from telegram import *
from telegram.ext import *
import tg_sql as sql
from warnings import filterwarnings
from telegram.warnings import PTBUserWarning
import json

filterwarnings(action="ignore", message=r".*CallbackQueryHandler",
category=PTBUserWarning)

(
    SELECTING_ACTION,
    SETUP_DATA,
    EDITING,
    AGE,
    HEIGHT,
    WEIGHT,
    INFO,
    TRAINS,
    ADD_TRAIN,
    REMOVE_TRAIN,
    UPLOAD_TRAINS,
    SHOW_TRAINS_PAGE,
    PAGE,
    BACK_BASE,
    FORWARD_BASE,
    DELETE_EXACT_TRAIN,
    LOAD_EXACT_TRAIN,
    ADDIT_INFO,
    TYPING,
    PROFILE,
    PROFILE_EDITING,
    PERSONAL_PLAN,
    SHOW_BASE,
    SELF_TRAINS_MENU,
    DELETE_TRAINS,
    SHOW_TRAINS,
    AUTHOR,
    TO_FEEDBACK,
    FEEDBACK,
    TO_MENU,
    BACK,
    STOP,
) = map(chr, range(32))

app = ApplicationBuilder().token("TG_TOKEN").build()

def find_train_by_number(number, data):
    i = 1
    for item in data:
        if i == number:
            return item
        i += 1

def setup_user_data(update, context):
    user_data = context.user_data
    if update.callback_query:
        data = sql.get_data_from_users(update.callback_query.from_user.id)
    else:
```

```

        data = sql.get_data_from_users(update.message.from_user.id)
        user_data[INFO] = {AGE: str(data['age']), WEIGHT: str(data['weight']),
HEIGHT: str(data['height'])}

        if data['add_info'] is None or not(data['add_info'] and
data['add_info'].strip()):
            user_data[INFO][ADDIT_INFO] = "Отсутствует"
        else:
            user_data[INFO][ADDIT_INFO] = data['add_info']

        if data['trains'] == '[]' or data['trains'] is None:
            user_data[INFO][TRAINS] = list()
        else:
            trains = json.loads(data['trains'])
            user_data[INFO][TRAINS] = trains

        user_data[INFO][UPLOAD_TRAINS] = {'Status':0}
        user_data[INFO][PAGE] = 1

    async def start(update, context):
        user_id = update.message.from_user.id

        text = "Привет, добро пожаловать в бота, позволяющего создать план
индивидуальной тренировки!"
        await update.message.reply_text(text=text)

        if not sql.is_user_in_users(user_id):
            context.user_data[INFO] = {AGE: 'None', HEIGHT: 'None', WEIGHT: 'None'}
            context.user_data[EDITING] = ''
            return await menu_of_data(update, context, "Вам необходимо заполнить
данные профиля: ")
        else:
            setup_user_data(update, context)
            context.user_data[BACK] = False
            if not sql.is_user_profile_full(user_id):
                context.user_data[EDITING] = ''
                return await menu_of_data(update, context, "Вам необходимо дополнить
данные профиля: ")
            else:
                context.user_data[EDITING] = None
                return await menu(update, context)

    async def menu_of_data(update, context, text="Выберите пункт редактирования"):
        user_data = context.user_data
        text = text
        media_photo = 'http://a1048263.xsph.ru/wp-
content/uploads/2024/11/image_menu.jpg'
        media = InputMediaPhoto(media=media_photo, caption=text)

        buttons = [
            [
                InlineKeyboardButton(text="Рост", callback_data=str(HEIGHT)),
                InlineKeyboardButton(text="Вес", callback_data=str(WEIGHT)),
            ],
            [
                InlineKeyboardButton(text="Возраст", callback_data=str(AGE)),
            ],
        ]

        if user_data[INFO][AGE] != 'None' and user_data[INFO][HEIGHT] != 'None' and
user_data[INFO][WEIGHT] != 'None':
            setup_user_data(update, context)
            buttons.append([InlineKeyboardButton(text="В меню",

```

```

callback_data=str(TO_MENU))],)

keyboard = InlineKeyboardMarkup(buttons)

if user_data[EDITING] is None:
    await update.callback_query.edit_message_media(media=media,
reply_markup=keyboard)
else:
    await update.message.reply_photo(
        photo=media_photo,
        caption=text,
        reply_markup=keyboard
    )

user_data[BACK] = True

return SETUP_DATA

async def editing(update, context):
    user_data = context.user_data
    if update.callback_query:
        user_data[EDITING] = update.callback_query.data

    button = [[InlineKeyboardButton(text="Назад", callback_data=str(TO_MENU))]]
    text = ""
    if user_data[EDITING] == AGE:
        text = "Введите информацию о возрасте: "
    elif user_data[EDITING] == HEIGHT:
        text = "Введите информацию о росте: "
    elif user_data[EDITING] == WEIGHT:
        text = "Введите информацию о весе: "
    elif user_data[EDITING] == ADDIT_INFO:
        text = "Введите желаемую заметку: "
    elif user_data[EDITING] == ADD_TRAIN:
        button.append([InlineKeyboardButton(text="Выбрать из базы",
switch_inline_query_current_chat="")])
        text = "Введите текст пункта: \n\n"
        text += "*Вы можете предложить добавить в базу тренировку\n"
        text += "*Об авторе -> Обратная связь\n"
    elif user_data[EDITING] == REMOVE_TRAIN:
        text = "Введите номер пункта: \n\n"
        i = 1
        for item in user_data[INFO][TRAINS]:
            text += str(i) + ". " + item + "\n"
            i += 1
    elif user_data[EDITING] == UPLOAD_TRAINS:
        stat = user_data[INFO][UPLOAD_TRAINS]
        if stat['Status'] == 1:
            text = "Введите имя вашего плана: "
        elif stat['Status'] == 2:
            text = "Введите описание вашего плана: "
    elif user_data[EDITING] == DELETE_TRAINS or user_data[EDITING] ==
SHOW_TRAINS:
        text = "Введите номер плана: \n\n"
        i = 1
        for item in user_data[INFO][SHOW_TRAINS_PAGE]:
            name = ''
            if user_data[INFO][SHOW_TRAINS]['status'] == 'base':
                name += " от "
            try:
                profile_user = await context.bot.get_chat(item['tg_id'])
                name += profile_user.first_name.capitalize()
            except:

```

```

        name += "Неизвестно"
        text += str(i) + ". " + item['train_name'] + name + "\n"
        i += 1
    reply = InlineKeyboardMarkup(button)

    media_photo = 'http://a1048263.xsph.ru/wp-
content/uploads/2024/11/image_menu.jpg'
    media = InputMediaPhoto(media=media_photo, caption=text)
    if update.callback_query:
        await update.callback_query.edit_message_media(media=media,
reply_markup=reply)
    else:
        await update.message.reply_photo(
            photo=media_photo,
            caption=text,
            reply_markup=reply
        )

    return TYPING

async def save_info(update, context):
    user_data = context.user_data
    user_data_edit = user_data[EDITING]
    user_text = update.message.text
    user_id = update.message.from_user.id

    if user_data_edit == AGE or user_data_edit == WEIGHT or user_data_edit ==
HEIGHT or user_data_edit == REMOVE_TRAIN or user_data_edit == DELETE_TRAINS or
user_data[EDITING] == SHOW_TRAINS:
        if not(user_text.isnumeric()):
            text = "Некоректные данные!\nПовторите попытку: "
            await update.message.reply_text(text=text)
            return TYPING
        else:
            user_text = int(user_text)

    if user_data_edit == AGE:
        if user_text <= 0 or user_text > 120:
            text = "Некоректные данные!\nПовторите попытку: "
            await update.message.reply_text(text=text)
            return TYPING
    elif user_data_edit == WEIGHT:
        if user_text <= 20 or user_text > 500:
            text = "Некоректные данные!\nПовторите попытку: "
            await update.message.reply_text(text=text)
            return TYPING
    elif user_data_edit == HEIGHT:
        if user_text <= 70 or user_text > 255:
            text = "Некоректные данные!\nПовторите попытку: "
            await update.message.reply_text(text=text)
            return TYPING
    elif user_data_edit == ADDIT_INFO or user_data_edit == ADD_TRAIN:
        if len(user_text) >=150:
            text = "Текст слишком длинный!\nПовторите попытку: "
            await update.message.reply_text(text=text)
            return TYPING
    elif user_data_edit == REMOVE_TRAIN:
        if int(user_text) > len(user_data[INFO][TRAINS]) or int(user_text) < 1:
            text = "Такого пункта не существует!\nПовторите попытку: "
            await update.message.reply_text(text=text)
            return TYPING
    elif user_data_edit == UPLOAD_TRAINS:
        if user_data[INFO][UPLOAD_TRAINS]['Status'] == 1:

```

```

        if sql.is_user_train_in_bd_by_name(user_id, user_text):
            text = "У вас уже есть план с таким именем!\nПовторите попытку: "
            await update.message.reply_text(text=text)
            return TYPING
        if len(user_text) >= 150:
            text = "Текст слишком длинный!\nПовторите попытку: "
            await update.message.reply_text(text=text)
            return TYPING
        elif user_data_edit == DELETE_TRAINS or user_data[EDITING] == SHOW_TRAINS:
            if not find_train_by_number(int(user_text),
user_data[INFO][SHOW_TRAINS_PAGE]) or int(user_text) < 1:
                text = "Такого пункта не существует!\nПовторите попытку: "
                await update.message.reply_text(text=text)
                return TYPING

        if user_data_edit == ADD_TRAIN:
            user_data[INFO][TRAINS].append(str(user_text))
        elif user_data_edit == REMOVE_TRAIN:
            user_data[INFO][TRAINS].pop(user_text - 1)
        elif user_data_edit == UPLOAD_TRAINS:
            stat = user_data[INFO][UPLOAD_TRAINS]
            if stat['Status'] == 1:
                stat['Name'] = str(user_text)
                stat['Status'] = 2
                return await editing(update, context)
            elif stat['Status'] == 2:
                stat['Descp'] = str(user_text)
        elif user_data_edit == DELETE_TRAINS:
            user_data[INFO][DELETE_TRAINS] = find_train_by_number(int(user_text),
user_data[INFO][SHOW_TRAINS_PAGE])['train_name']
        elif user_data[EDITING] == SHOW_TRAINS:
            user_data[INFO][SHOW_TRAINS]['train'] =
find_train_by_number(int(user_text), user_data[INFO][SHOW_TRAINS_PAGE])
        else:
            user_data[INFO][user_data_edit] = str(user_text)
            saving(update, context)

        if user_data_edit == ADDIT_INFO or user_data_edit == ADD_TRAIN or
user_data_edit == REMOVE_TRAIN:
            user_data[BACK] = False
            return await show_plan(update, context)
        elif user_data_edit == UPLOAD_TRAINS or user_data_edit == DELETE_TRAINS:
            user_data[BACK] = False
            return await self.trains_menu(update, context)
        elif user_data[EDITING] == SHOW_TRAINS:
            return await show_exact_trains(update, context)

    return await menu_of_data(update, context, "Продолжаем заполнение: ")

def saving(update, context):
    user_data = context.user_data
    user_id = update.message.from_user.id

    if user_data[EDITING] == AGE:
        sql.set_user_age(user_id, user_data[INFO][AGE])
    elif user_data[EDITING] == WEIGHT:
        sql.set_user_weight(user_id, user_data[INFO][WEIGHT])
    elif user_data[EDITING] == HEIGHT:
        sql.set_user_height(user_id, user_data[INFO][HEIGHT])

```



```

        elif user_data[EDITING] == ADD_TRAIN or user_data[EDITING] == REMOVE_TRAIN:
            sql.set_user_trains(user_id, json.dumps(user_data[INFO][TRAINS],
ensure_ascii=False))
        elif user_data[EDITING] == ADDIT_INFO:
            sql.set_user_info(user_id, user_data[INFO][ADDIT_INFO])
        elif user_data[EDITING] == UPLOAD_TRAINS and
user_data[INFO][UPLOAD_TRAINS]['Status'] == 2:
            sql.upload_user_train(user_id, user_data[INFO][UPLOAD_TRAINS]['Name'],
sql.get_data_from_users(user_id)['trains'],
user_data[INFO][UPLOAD_TRAINS]['Descp'])
            user_data[INFO][UPLOAD_TRAINS]['Status'] = 0
        elif user_data[EDITING] == DELETE_TRAINS:
            sql.delete_user_train_by_name(user_id, user_data[INFO][DELETE_TRAINS])

async def menu(update, context):
    context.user_data[INFO][SHOW_TRAINS_PAGE] = list()
    context.user_data[INFO][PAGE] = 1
    context.user_data[EDITING] = None

    text = "Вы можете выбрать своё дальнейшее действие: "
    media_photo = 'http://a1048263.xsph.ru/wp-
content/uploads/2024/11/image_menu.jpg'
    media = InputMediaPhoto(media=media_photo, caption=text)

    buttons = [
        [
            #InlineKeyboardButton(text="Напоминания",
callback_data=str(REMINDERS)),
            InlineKeyboardButton(text="План тренировок",
callback_data=str(PERSONAL_PLAN)),
        ],
        [
            InlineKeyboardButton(text="База готовых планов",
callback_data=str(SHOW_BASE)),
        ],
        [
            InlineKeyboardButton(text="Профиль", callback_data=str(PROFILE)),
        ],
        [
            InlineKeyboardButton(text="Об авторе", callback_data=str(AUTHOR)),
        ]
    ]
    keyboard = InlineKeyboardMarkup(buttons)

    if context.user_data[BACK]:
        await update.callback_query.edit_message_media(media=media,
reply_markup=keyboard)
    else:
        await update.message.reply_photo(
            photo=media_photo,
            caption=text,
            reply_markup=keyboard
        )

    context.user_data[BACK] = True
    return SELECTING_ACTION

async def author(update, context):
    text = "Автор : G3RKA \nДизайнер: SickoSasha\nСпонсор: The Frokich \nВерсия:
beta v1"
    media_photo = 'http://a1048263.xsph.ru/wp-
content/uploads/2024/11/image_author.png'

```

```

media = InputMediaPhoto(media=media_photo, caption=text)

buttons = [
    [InlineKeyboardButton(text="Обратная связь",
callback_data=str(TO_FEEDBACK))],
    [InlineKeyboardButton(text="Назад", callback_data=str(TO_MENU))]]
]
keyboard = InlineKeyboardMarkup(buttons)

await update.callback_query.edit_message_media(media=media,
reply_markup=keyboard)

context.user_data[BACK] = True

return SELECTING_ACTION

async def feedback(update, context):
    text = "Добро пожаловать в обратную связь!\nВведите сообщение, которое вы
хотите отправить автору."
    media_photo = 'http://a1048263.xsph.ru/wp-
content/uploads/2024/11/image_feedback.png'
    media = InputMediaPhoto(media=media_photo, caption=text)

    await update.callback_query.edit_message_media(media=media)

    context.user_data[BACK] = False

    return FEEDBACK

async def send_feed(update, context):
    text = update.message.text
    print("Отправлено сообщение по обратной связи:\n", text)

    text = "Получено новое сообщение по обратной связи от @" +
update.message.from_user.username + ':\n'+ text
    await context.bot.send_message(text=text, chat_id=605787598)

    return await menu(update, context)

async def profile(update, context):
    user_data = context.user_data

    text = "Профиль " + update.callback_query.from_user.username
    media_photo = await update.callback_query.from_user.get_profile_photos()
    media_photo = media_photo.photos[0][0].file_id

    buttons = [
        [InlineKeyboardButton(text="Редактировать",
callback_data=str(PROFILE_EDITING))],
        [InlineKeyboardButton(text="Назад", callback_data=str(TO_MENU))]]
    ]
    keyboard = InlineKeyboardMarkup(buttons)

    text += "\nВозраст: " + user_data[INFO][AGE]
    text += "\nРост: " + user_data[INFO][HEIGHT]
    text += "\nВес: " + user_data[INFO][WEIGHT]

    media = InputMediaPhoto(media=media_photo, caption=text)

    await update.callback_query.edit_message_media(media=media,
reply_markup=keyboard)

    context.user_data[BACK] = True

```

```

        return SELECTING_ACTION

async def base_train(update, context):
    user_data = context.user_data
    if update.callback_query:
        if update.callback_query.data == FORWARD_BASE:
            user_data[BACK] = True
            user_data[INFO][PAGE] += 1
        elif update.callback_query.data == BACK_BASE:
            user_data[BACK] = True
            user_data[INFO][PAGE] -= 1

    page = user_data[INFO][PAGE]
    user_data[INFO][SHOW_TRAINS_PAGE] = list()
    user_data[INFO][SHOW_TRAINS] = {'status': 'base'}
    text = "Открытая база планов тренировок от других пользователей.\n\n"
    media_photo = 'http://a1048263.xsph.ru/wp-
content/uploads/2024/11/image_base.jpg'

    data = sql.get_all_users_trains_by_page()
    data_to_show = data[(10 * (page-1)): (10 * page)]

    text += "Список планов:\n\n"
    if len(data_to_show) == 0:
        text += "Отсутствует\n"
    else:
        i = 1
        for item in data_to_show:
            try:
                profile_user = await context.bot.get_chat(item['tg_id'])
                name = profile_user.first_name.capitalize()
            except:
                name = "Неизвестно"
            user_data[INFO][SHOW_TRAINS_PAGE].append(item)
            text += str(i) + ". " + item['train_name'] + " от " + name + "\n"
            i += 1
            text += "\n"

    move_buttons = list()
    if page > 1:
        move_buttons.append(InlineKeyboardButton(text="Назад",
callback_data=str(BACK_BASE)))
    if len(data) - (10*page) > 0:
        move_buttons.append(InlineKeyboardButton(text="Вперёд",
callback_data=str(FORWARD_BASE)))

    upload_buttons = [InlineKeyboardButton(text="Мои планы",
callback_data=str(SELF_TRAINS_MENU))]

    buttons = [
        move_buttons,
        upload_buttons,
        [(InlineKeyboardButton(text="Просмотреть пункт",
callback_data=str(SHOW_TRAINS)))],
        [InlineKeyboardButton(text="В меню", callback_data=str(TO_MENU))]
    ]

    keyboard = InlineKeyboardMarkup(buttons)

    media = InputMediaPhoto(media=media_photo, caption=text)

    if context.user_data[BACK]:

```

```

        await update.callback_query.edit_message_media(media=media,
reply_markup=keyboard)
    else:
        await update.message.reply_photo(
            photo=media_photo,
            caption=text,
            reply_markup=keyboard
        )

    context.user_data[BACK] = True

    return SELECTING_ACTION

async def self_trains_menu(update, context):
    if update.callback_query:
        user_id = update.callback_query.from_user.id
    else:
        user_id = update.message.from_user.id
    user_data = context.user_data
    user_data[INFO][SHOW_TRAINS] = {'status': 'self'}
    text = "Список опубликованных вами планов.\n\n"
    media_photo = 'http://a1048263.xsph.ru/wp-
content/uploads/2024/11/image_base.jpg'

    user_data[INFO][SHOW_TRAINS_PAGE] = list()
    adding_buttons = list()
    show_buttons = list()
    if len(sql.get_all_user_train(user_id)) < 10:
        adding_buttons.append(InlineKeyboardButton(text="Загрузить текущий
план", callback_data=str(UPLOAD_TRAINS)))
    if len(sql.get_all_user_train(user_id)) > 0:
        adding_buttons.append(InlineKeyboardButton(text="Удалить план",
callback_data=str(DELETE_TRAINS)))
        show_buttons = [(InlineKeyboardButton(text="Просмотреть пункт",
callback_data=str(SHOW_TRAINS)))]

    buttons = [
        adding_buttons,
        show_buttons,
        [InlineKeyboardButton(text="В меню", callback_data=str(TO_MENU))],
    ]

    keyboard = InlineKeyboardMarkup(buttons)

    text += "Список планов:\n\n"
    if len(sql.get_all_user_train(user_id)) == 0:
        text += "Отсутствует\n"
    else:
        i = 1
        for item in sql.get_all_user_train(user_id):
            user_data[INFO][SHOW_TRAINS_PAGE].append(item)
            text += str(i) + ". " + item['train_name'] + "\n"
            i += 1

    text += "\n"

    text += "Лимит общего количества планов: 10\n"

    media = InputMediaPhoto(media=media_photo, caption=text)

    if context.user_data[BACK]:
        await update.callback_query.edit_message_media(media=media,

```

```

reply_markup=keyboard)
    else:
        await update.message.reply_photo(
            photo=media_photo,
            caption=text,
            reply_markup=keyboard
        )

        context.user_data[BACK] = True
        return SELECTING_ACTION

async def show_exact_trains(update, context):
    user_data = context.user_data
    text = "Информация о плане тренировки.\n\n"
    media_photo = 'http://a1048263.xsph.ru/wp-
content/uploads/2024/11/image_base.jpg'
    buttons = list()

    train = user_data[INFO][SHOW_TRAINS]['train']

    try:
        profile_user = await context.bot.get_chat(train['tg_id'])
        name = profile_user.first_name.capitalize()
    except:
        name = "Неизвестно"

    text += "Имя: " + train["train_name"] + "\n"

    text += "Описание: " + train["descp"] + "\n"

    text += "План: " + "\n"
    i = 1
    for item in json.loads(train["trains"]):
        text += str(i) + ". " + item + "\n"
        i += 1
    text += "\n"
    text += "Создатель: " + name

    text += "\n\n"

    if int(train['tg_id']) == update.message.from_user.id:
        buttons.append([InlineKeyboardButton(text="Удалить план",
callback_data=str(DELETE_EXACT_TRAIN))])

    text += "*При загрузке плана Ваш текущий план исчезнет!"
    buttons.append([InlineKeyboardButton(text="Загрузить план себе",
callback_data=str(LOAD_EXACT_TRAIN))])
    buttons.append([InlineKeyboardButton(text="В меню",
callback_data=str(TO_MENU))])

    keyboard = InlineKeyboardMarkup(buttons)

    await update.message.reply_photo(
        photo=media_photo,
        caption=text,
        reply_markup=keyboard
    )

    context.user_data[BACK] = True
    return SELECTING_ACTION

async def delete_exact_train(update, context):

```

```

user_data = context.user_data
user_id = update.callback_query.from_user.id

train = user_data[INFO][SHOW_TRAINS]['train']

sql.delete_user_train_by_name(user_id, train['train_name'])

context.user_data[BACK] = True
if user_data[INFO][SHOW_TRAINS]['status'] == 'self':
    return await self_trains_menu(update, context)
elif user_data[INFO][SHOW_TRAINS]['status'] == 'base':
    return await base_train(update, context)

async def load_exact_train(update, context):
    user_data = context.user_data
    user_id = update.callback_query.from_user.id
    train = user_data[INFO][SHOW_TRAINS]['train']
    trains = json.loads(train['trains'])

    user_data[INFO][TRAINS] = trains
    sql.set_user_trains(user_id, json.dumps(user_data[INFO][TRAINS],
ensure_ascii=False))

    user_data[BACK] = True
    return await show_plan(update, context)

async def show_plan(update, context):
    user_data = context.user_data

    text = "Ваш личный план тренировок: \n\n"
    media_photo = 'http://a1048263.xsph.ru/wp-
content/uploads/2024/11/image_plan.jpg'

    adding_buttons = list()

    if len(user_data[INFO][TRAINS]) <= 15:
        adding_buttons.append(InlineKeyboardButton(text="Добавить пункт плана",
callback_data=str(ADD_TRAIN)))
    if len(user_data[INFO][TRAINS]) > 0:
        adding_buttons.append(InlineKeyboardButton(text="Удалить пункт плана",
callback_data=str(REMOVE_TRAIN)))

    buttons = [
        adding_buttons,
        [InlineKeyboardButton(text="Редактировать заметку",
callback_data=str(ADDIT_INFO))],
        [InlineKeyboardButton(text="Назад", callback_data=str(TO_MENU))]
    ]
    keyboard = InlineKeyboardMarkup(buttons)

    text += "Список упражнений:\n"
    if len(user_data[INFO][TRAINS]) == 0:
        text += "Отсутствует\n"
    else:
        i = 1
        for item in user_data[INFO][TRAINS]:
            text += str(i) + ". " + item + "\n"
            i += 1

    text += "\n"

```

```

text += "Личная заметка:\n"
text += user_data[INFO][ADDIT_INFO] + "\n\n"

text += "*Максимум пунктов: 15\n"

media = InputMediaPhoto(media=media_photo, caption=text)

if context.user_data[BACK]:
    await update.callback_query.edit_message_media(media=media,
reply_markup=keyboard)
else:
    await update.message.reply_photo(
        photo=media_photo,
        caption=text,
        reply_markup=keyboard
    )

context.user_data[BACK] = True

return SELECTING_ACTION

async def upload_trains(update, context):
    user_data = context.user_data
    user_data[INFO][UPLOAD_TRAINS]['Status'] = 1
    return await editing(update, context)

async def stop(update, context):
    text = "ОШИБКА. ТЫ НЕ ДОЛЖЕН ЗДЕСЬ НАХОДИТЬСЯ"
    media_photo =
'https://media.discordapp.net/attachments/4065075556954177548/1296119098568212572
/image.png?ex=671860b9&is=67170f39&hm=1cf365ab667e6b1cbf5e910393b5c7cb60ec81be69
1aebdff693c44da75ca8dc&format=webp&quality=lossless&width=364&height=350&'
    media = InputMediaPhoto(media=media_photo, caption=text)
    await update.callback_query.edit_message_media(media=media)

    return STOP

async def inline_query_res(update, context):
    query = update.inline_query.query

    data = sql.get_all_trains()
    results = list()

    i = 1
    for item in data:
        training = item['trains']
        if str(query).lower() in training.lower() or len(query.strip()) == 0:
            results.append(InlineQueryResultArticle(
                id=str(i),
                title=training,
                input_message_content=InputTextMessageContent(training),
            ))
            i += 1
        else:
            continue

    await update.inline_query.answer(results)

def main():
    selection_handlers = [
        CallbackQueryHandler(profile, pattern="^" + str(PROFILE) + "$"),
        CallbackQueryHandler(author, pattern="^" + str(AUTHOR) + "$"),

```

```

        CallbackQueryHandler(feedback, pattern="^" + str(TO_FEEDBACK) + "$"),
        CallbackQueryHandler(base_train, pattern="^" + str(SHOW_BASE) + "$|^" +
str(FORWARD_BASE) + "$|^" + str(BACK_BASE) + "$"),
        CallbackQueryHandler(self_trains_menu, pattern="^" +
str(SELF_TRAINS_MENU) + "$"),
        CallbackQueryHandler(show_plan, pattern="^" + str(PERSONAL_PLAN) + "$"),
        CallbackQueryHandler(upload_trains, pattern="^" + str(UPLOAD_TRAINS) +
"$"),
        CallbackQueryHandler(delete_exact_train, pattern="^" +
str(DELETE_EXACT_TRAIN) + "$"),
        CallbackQueryHandler(load_exact_train, pattern="^" +
str(LOAD_EXACT_TRAIN) + "$"),
        CallbackQueryHandler(editing, pattern="^" + str(ADDIT_INFO) + "$|^"+
str(ADD_TRAIN)+ "$|^" + str(REMOVE_TRAIN) + "$|^" + str(DELETE_TRAINS) + "$|^" +
str(SHOW_TRAINS) + "$"),
        CallbackQueryHandler(menu_of_data, pattern="^" + str(PROFILE_EDITING) +
"$"),
        CallbackQueryHandler(menu, pattern="^" + str(TO_MENU) + "$"),
    ]

    selection_data = [
        CallbackQueryHandler(editing, pattern="^" + str(AGE) + "$|^" +
str(HEIGHT)+ "$|^" + str(WEIGHT) + "$"),
        CallbackQueryHandler(menu, pattern="^" + str(TO_MENU) + "$"),
    ]

    conv_handler = ConversationHandler(
        entry_points=[CommandHandler("start", start)],
        states={
            FEEDBACK: [MessageHandler(filters=filters.TEXT & ~filters.COMMAND,
callback=send_feed)],
            TYPING: [
                CallbackQueryHandler(menu, pattern="^" + str(TO_MENU) + "$"),
                MessageHandler(filters=filters.TEXT & ~filters.COMMAND,
callback=save_info)
            ],
            SELECTING_ACTION: selection_handlers,
            SETUP_DATA: selection_data,
            STOP: [CommandHandler("start", start)],
        },
        fallbacks=[
            CallbackQueryHandler(stop),
        ],
        allow_reentry = True,
    )

    app.add_handler(conv_handler)
    app.add_handler(InlineQueryHandler(inline_query_res), 1)

    if not sql.check_server():
        print("ОШИБКА: БАЗА НЕ ПОДКЛЮЧЕНА")
        print("Остановка.")
        app.stop_running()
    else:
        print("Бот успешно запущен!")
        app.run_polling()

if __name__ == "__main__":
    main()

```

Файл tg_sql.py


```

import mysql.connector as sql
from mysql.connector import errorcode

server = None

config = {
    'user': 'name',
    'password': 'password',
    'host': 'ip',
    'database': 'database',
    'raise_on_warnings': True
}

try:
    server = sql.connect(**config)
except sql.Error as err:
    print("БАЗА НЕ ПОДКЛЮЧЕНА")
    if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
        print("Данные неверны!")
    elif err.errno == errorcode.ER_BAD_DB_ERROR:
        print("БД не существует!")
    else:
        print(err)

def check_server():
    if not server:
        return None
    if not server.is_connected():
        return None
    return True

def check_and_retry_connection():
    if not server.is_connected():
        server.reconnect(attempts=5, delay=2)

def is_user_train_in_bd_by_name(tg_id, name):
    data = get_all_user_train(tg_id)
    for items in data:
        if items['train_name'] == name:
            return True
    return False

def is_user_in_users(tg_id):
    data = get_data_from_users(tg_id)
    if not data:
        return False
    else:
        return True

def is_user_profile_contain_trains(tg_id):
    data = get_data_from_users(tg_id)
    if data["trains"] is None:
        return False
    return True

def is_user_profile_full(tg_id):
    data = get_data_from_users(tg_id)
    if data["height"] is None or data["weight"] is None or data["age"] is None:
        return False
    return True

def get_data_from_users(tg_id):

```

```

    check_and_retry_connection()
    server.commit()
    curs = server.cursor(dictionary=True)
    query = "SELECT * FROM users WHERE tg_id = %s"
    curs.execute(query, (str(tg_id),))
    data = curs.fetchone()
    curs.close()

    return data

def set_user_age(tg_id, age):
    check_and_retry_connection()
    curs = server.cursor(buffered=True)
    if not is_user_in_users(tg_id):
        query = "INSERT INTO users (tg_id, age) VALUES (%s, %s)"
        curs.execute(query, (str(tg_id), int(age)))
    else:
        query = "UPDATE users SET age = %s WHERE tg_id = %s"
        curs.execute(query, (int(age), str(tg_id)))
    server.commit()
    curs.close()

def set_user_weight(tg_id, weight):
    check_and_retry_connection()
    curs = server.cursor(buffered=True)
    if not is_user_in_users(tg_id):
        query = "INSERT INTO users (tg_id, weight) VALUES (%s, %s)"
        curs.execute(query, (str(tg_id), int(weight)))
    else:
        query = "UPDATE users SET weight = %s WHERE tg_id = %s"
        curs.execute(query, (int(weight), str(tg_id)))
    server.commit()
    curs.close()

def set_user_height(tg_id, height):
    check_and_retry_connection()
    curs = server.cursor(buffered=True)
    if not is_user_in_users(tg_id):
        query = "INSERT INTO users (tg_id, height) VALUES (%s, %s)"
        curs.execute(query, (str(tg_id), int(height)))
    else:
        query = "UPDATE users SET height = %s WHERE tg_id = %s"
        curs.execute(query, (int(height), str(tg_id)))
    server.commit()
    curs.close()

def set_user_info(tg_id, add_info):
    check_and_retry_connection()
    curs = server.cursor(buffered=True)
    if not is_user_in_users(tg_id):
        query = "INSERT INTO users (tg_id, add_info) VALUES (%s, %s)"
        curs.execute(query, (str(tg_id), add_info))
    else:
        query = "UPDATE users SET add_info = %s WHERE tg_id = %s"
        curs.execute(query, (add_info, str(tg_id)))
    server.commit()
    curs.close()

def set_user_trains(tg_id, trains):
    check_and_retry_connection()
    curs = server.cursor(buffered=True)

```

```

if not is_user_in_users(tg_id):
    query = "INSERT INTO users (tg_id, trains) VALUES (%s, %s)"
    curs.execute(query, (str(tg_id), trains))
else:
    query = "UPDATE users SET trains = %s WHERE tg_id = %s"
    curs.execute(query, (trains, str(tg_id)))
server.commit()
curs.close()

def get_all_users_trains_by_page():
    check_and_retry_connection()
    server.commit()
    curs = server.cursor(dictionary=True)
    query = "SELECT * FROM trains_users"
    curs.execute(query)
    data = curs.fetchall()
    curs.close()

    return data

def get_all_user_train(tg_id):
    check_and_retry_connection()
    server.commit()
    curs = server.cursor(dictionary=True)

    query = "SELECT * FROM `trains_users` WHERE tg_id = %s"
    curs.execute(query, (str(tg_id),))
    data = curs.fetchall()

    curs.close()

    return data

def upload_user_train(tg_id, name, trains, desc):
    check_and_retry_connection()
    curs = server.cursor(buffered=True)

    query = "INSERT INTO trains_users(`tg_id`, `train_name`, `trains`, `descp`)
VALUES (%s, %s, %s, %s)"
    curs.execute(query, (str(tg_id), name, trains, desc))

    server.commit()
    curs.close()

def delete_user_train_by_name(tg_id, name):
    check_and_retry_connection()
    curs = server.cursor(buffered=True)

    query = "DELETE FROM `trains_users` WHERE `tg_id` = %s AND `train_name` = %s
LIMIT 1"

    curs.execute(query, (str(tg_id), name))

    server.commit()
    curs.close()

def get_user_train_by_name(tg_id, name):
    check_and_retry_connection()
    server.commit()
    curs = server.cursor(dictionary=True)

    query = "SELECT * FROM `trains_users` WHERE `tg_id` = %s AND `train_name` =
%s"

```

```
    curs.execute(query, (str(tg_id), name))
    data = curs.fetchall()

    curs.close()

    return data

def get_all_trains():
    check_and_retry_connection()
    server.commit()
    curs = server.cursor(dictionary=True)

    query = "SELECT * FROM `trains`"
    curs.execute(query)
    data = curs.fetchall()

    curs.close()

    return data
```