Московский государственный технический

университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по домашнему заданию.

Выполнил:

студент группы ИУ5-33Б

Абитов М.Р.

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Гапанюк Ю.Е.

Подпись и дата:

Москва, 2023 г.

Общее описание проекта.

Проект, реализованный в качестве домашнего задания по дисциплине, представляет собой разработанного на языке Python Telegram – бота с использованием библиотеки Telebot, основной темой и идеей которого является работа с базой данных на основе ORM peewee:

Пользователь взаимодействует с другими зарегистрированными пользователями, а вся информация о пользователях хранится в базе данных.

Тема и актуальность проекта.

Данная тема была выбрана в связи с высокой ролью операций над различными валютами в жизни человека и необходимостью совершать переводы между людьми, при этом чтобы была возможность отслеживания истории транзакций, а также поддержка конвертации валют.

Функционал проекта.

Созданный Telegram – бот позволяет производить конвертацию нескольких указанных валют и предоставлять возможность просмотра всех пользователей БД, перевод между пользователями, смену валюты персонального счета, а также просмотр истории транзакций. Еще добавлено автосохранение базы данных результатов каждой и сессий работы данного бота и последующий анализ данной базы данных.

Выбор языка программирования.

Для создания проекта был выбран язык Python с целью получения возможностей изучения создания Telegram – ботов, работы асинхронных функций и взаимодействия с API.

Код проекта.

models.py

```
from peewee import *
from datetime import datetime
import os
import shutil

db = SqliteDatabase('data.db')


class BaseModel(Model):
    class Meta:
        database = db


class Currency(BaseModel):
    code = CharField(unique=True)
    exchange_rate = DecimalField(max_digits=15, decimal_places=6, default=1.0)
    is_base_currency = BooleanField(default=False)


class Account(BaseModel):
    user_id = PrimaryKeyField()
    name = CharField()
    balance = DecimalField(max_digits=15, decimal_places=2)
    trader = BooleanField()
    currency = ForeignKeyField(Currency, backref="accounts")


class Transaction(BaseModel):
    sender = ForeignKeyField(
        Account, backref="sent_transactions", column_name='sender_id')
    receiver = ForeignKeyField(
        Account, backref="received_transactions", column_name='receiver_id')
    amount = DecimalField(max_digits=15, decimal_places=2)
    currency = CharField()
    timestamp = DateTimeField(default=datetime.now)


class TransferState(BaseModel):
    user_id = IntegerField(unique=True)  # ID пользователя в Telegram
    step = CharField()  # Текущий шаг (например, 'awaiting_receiver_id',
'awaiting_amount')
    receiver_id = IntegerField(null=True)  # ID получателя перевода
```

```
    amount = DecimalField(max_digits=15, decimal_places=2,
                          null=True)  # Сумма перевода


class TransactionHistory(BaseModel):
    transaction = ForeignKeyField(
        Transaction, backref="history", column_name='transaction_id')
    message = CharField()  # Описание транзакции (например, "Перевод от Петра к
Ольге")


def initialize_database():
    db.connect()

    # Путь к текущей базе данных
    current_db_path = db.database
    # Путь к копии базы данных
    backup_db_path = f"backup_{datetime.now().strftime('%Y-%m-%d_%H-%M-%S')}.db"

    # Создание копии базы данных
    if os.path.exists(current_db_path):
        shutil.copyfile(current_db_path, backup_db_path)
        print(f"База данных скопирована: {backup_db_path}")

    # Удаление и создание новых таблиц
    db.drop_tables([Account, Transaction, Currency,
                    TransactionHistory], safe=True)
    db.create_tables([Account, Transaction, Currency,
                      TransactionHistory], safe=True)

    print("База данных инициализирована.")


def convert_currency(amount, from_currency, to_currency):
    try:
        from_currency_obj = Currency.get(Currency.code == from_currency)
        to_currency_obj = Currency.get(Currency.code == to_currency)
    except DoesNotExist:
        print(f"Валюта не найдена.")
        return amount

    if from_currency_obj.is_base_currency:
        converted_amount = amount * to_currency_obj.exchange_rate
    elif to_currency_obj.is_base_currency:
        converted_amount = amount / from_currency_obj.exchange_rate
    else:
        converted_amount = amount * \
            (1 / from_currency_obj.exchange_rate) * to_currency_obj.exchange_rate
```

```
    return converted_amount


def transfer_money(sender, receiver, amount):
    if sender.balance >= amount:
        converted_amount = convert_currency(
            amount, sender.currency.code, receiver.currency.code)
        sender.balance -= amount
        receiver.balance += converted_amount
        sender.save()
        receiver.save()
        transaction = Transaction.create(
            sender=sender, receiver=receiver,
amount=amount,  currency=sender.currency.code)
        TransactionHistory.create(
            transaction=transaction, message=f"Перевод от {sender.name} к
{receiver.name}")
        print(
            f"Перевод {amount: .2f} {sender.currency.code} от {sender.name} к
{receiver.name} выполнен успешно. ({converted_amount: .2f}
{receiver.currency.code})")
    else:
        print("Недостаточно средств для выполнения перевода.")


def display_all_accounts():
    accounts = Account.select()
    print("Список всех аккаунтов:")
    for account in accounts:
        print(
            f"ID: {account.user_id}, Имя: {account.name}, Баланс: {account.balance:
.2f} {account.currency.code}, Trader: {account.trader}")


def change_account_currency(account, new_currency):
    try:
        new_currency_obj = Currency.get(Currency.code == new_currency)
    except DoesNotExist:
        print(f"Валюта с кодом {new_currency} не найдена.")
        return

    if account.currency == new_currency_obj:
        print(f"Счет уже хранится в валюте {new_currency_obj.code}.")
        return

    converted_balance = convert_currency(
        account.balance, account.currency.code, new_currency_obj.code)

    account.balance = converted_balance
```

```
    account.currency = new_currency_obj
    account.save()

    print(
        f"Валюта счета успешно изменена на {new_currency_obj.code}. Новый баланс:
{account.balance: .2f} {new_currency_obj.code}")


def create_currency(code, exchange_rate=1.0, is_base_currency=False):
    return Currency.create(code=code, exchange_rate=exchange_rate,
is_base_currency=is_base_currency)


def display_transaction_history(account):
    print(f"История транзакций для аккаунта {account.name}:")
    transactions = Transaction.select().where(
        (Transaction.sender == account) | (Transaction.receiver == account)
    )
    for transaction in transactions:
        sender_name = transaction.sender.name if transaction.sender else "N/A"
        receiver_name = transaction.receiver.name if transaction.receiver else
"N/A"
        history_entry = TransactionHistory.get(
            TransactionHistory.transaction == transaction)
        print(
            f"ID: {transaction.id}, Отправитель: {sender_name}, Сумма:
{transaction.amount}, Описание: {history_entry.message}"
        )
```

test.py

```
import telebot
from telebot import types
from models import *
from decimal import Decimal

initialize_database()

# Добавим валюты
usd = create_currency("USD")
eur = create_currency("EUR", exchange_rate=0.85)
rub = create_currency("RUB", exchange_rate=65)  # Пример с рублями

# Создадим аккаунты
petr = Account.create(name='Пётр', balance=1000, trader=False, currency=usd)
olga = Account.create(name='Ольга', balance=1200, trader=False, currency=usd)
grig = Account.create(name='Григорий', balance=1500,
```

```
                      trader=False, currency=eur)

bot = telebot.TeleBot('6966266455:AAHoz_2aDK7o040xkvLjDAY4zJ4cOJj5V0I')


# Словарь для хранения данных о переводе
transfer_data = {}


@bot.message_handler(commands=['start'])
def start(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    btn1 = types.KeyboardButton("Регистрация")
    btn2 = types.KeyboardButton("Список пользователей")
    btn3 = types.KeyboardButton("Перевести деньги")
    btn4 = types.KeyboardButton("Смена валюты")
    btn5 = types.KeyboardButton("История транзакций")
    markup.add(btn1, btn2, btn3, btn4, btn5)

    bot.send_message(message.from_user.id,
                     "Выберите опцию:", reply_markup=markup)


def register_user(user_id, name):
    # Проверяем, существует ли пользователь
    existing_account = Account.select().where(Account.user_id == user_id)
    if existing_account.exists():
        return "Вы уже зарегистрированы."
    else:
        # Добавляем нового пользователя с начальным балансом 100 USD
        usd_currency = Currency.get(Currency.code == "USD")
        Account.create(user_id=user_id, name=name, balance=100.0,
                       trader=False, currency=usd_currency)
        return "Регистрация успешно завершена!"


@bot.message_handler(func=lambda message: message.text == "Регистрация")
def ask_for_name(message):
    msg = bot.send_message(message.from_user.id,
                           "Введите ваше имя для регистрации:")
    bot.register_next_step_handler(msg, process_name_step)


def process_name_step(message):
    try:
        user_id = message.from_user.id
        name = message.text
        response = register_user(user_id, name)
        bot.send_message(message.from_user.id, response)
```

```
    except Exception as e:
        bot.reply_to(message, 'Произошла ошибка.')


@bot.message_handler(func=lambda message: message.text == "Список пользователей")
def handle_user_list(message):
    accounts = Account.select()
    response = "Список всех пользователей:\n\n"
    for account in accounts:
        response += f"ID: {account.user_id}, Имя: {account.name}, Баланс:
{account.balance: .2f} {account.currency.code}\n"
    bot.send_message(message.from_user.id, response)


@bot.message_handler(func=lambda message: message.text == "Перевести деньги")
def ask_recipient_id(message):
    msg = bot.send_message(
        message.from_user.id, "Введите ID пользователя, которому хотите перевести
деньги:")
    bot.register_next_step_handler(msg, process_transfer_step)


def process_transfer_step(message):
    try:
        recipient_id = int(message.text)
        transfer_data[message.from_user.id] = {'recipient_id': recipient_id}
        msg = bot.send_message(message.from_user.id, 'Введите сумму перевода:')
        bot.register_next_step_handler(msg, process_amount_step)
    except ValueError:
        bot.send_message(message.from_user.id,
                         'Пожалуйста, введите корректный ID.')


def process_amount_step(message):
    try:
        amount = Decimal(message.text)
        sender_id = message.from_user.id
        recipient_id = transfer_data[sender_id]['recipient_id']

        # Извлечение объектов аккаунта из базы данных
        sender = Account.get_or_none(Account.user_id == sender_id)
        recipient = Account.get_or_none(Account.user_id == recipient_id)

        if sender is None or recipient is None:
            bot.send_message(message.from_user.id,
                             'Отправитель или получатель не найден.')
            return

        # Вызов функции перевода с объектами аккаунта
```

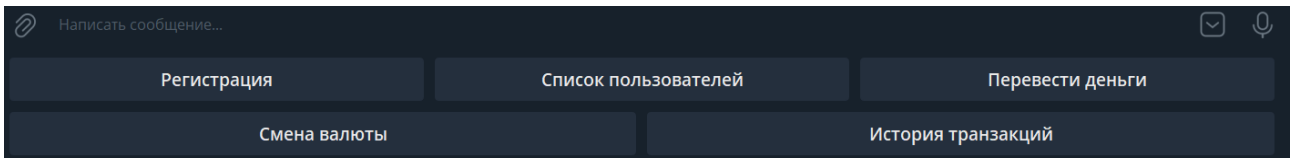
```
        transfer_money(sender, recipient, amount)
        bot.send_message(message.from_user.id, 'Перевод выполнен успешно.')
    except ValueError:
        bot.send_message(message.from_user.id,
                         'Пожалуйста, введите корректную сумму.')
    except Exception as e:
        bot.send_message(message.from_user.id, f'Произошла ошибка: {e}')


@bot.message_handler(func=lambda message: message.text == "Смена валюты")
def change_currency_menu(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    currencies = Currency.select()
    for currency in currencies:
        markup.add(types.KeyboardButton(currency.code))
    markup.add(types.KeyboardButton("Вернуться назад"))
    bot.send_message(message.from_user.id,
                     "Выберите валюту:", reply_markup=markup)


@bot.message_handler(func=lambda message: message.text in [currency.code for
currency in Currency.select()])
def handle_currency_change(message):
    new_currency_code = message.text
    user_id = message.from_user.id
    account = Account.get_or_none(Account.user_id == user_id)

    if account:
        old_balance, old_currency_code = account.balance, account.currency.code
        new_currency = Currency.get(Currency.code == new_currency_code)
        account.balance = convert_currency(
            old_balance, account.currency, new_currency)
        account.currency = new_currency
        account.save()

        response = f"Ваш баланс был переведен из {old_balance: .2f}
{old_currency_code} в {account.balance: .2f} {new_currency_code}."
        bot.send_message(user_id, response)
    else:
        bot.send_message(user_id, "Ваш аккаунт не найден.")

    start(message)  # Возвращаемся в основное меню


@bot.message_handler(func=lambda message: message.text == "Вернуться назад")
def back_to_main_menu(message):
    start(message)
```

```
def convert_currency(amount, from_currency, to_currency):
    return amount * (to_currency.exchange_rate / from_currency.exchange_rate)


@bot.message_handler(func=lambda message: message.text == "История транзакций")
def show_transaction_history(message):
    user_id = message.from_user.id
    account = Account.get_or_none(Account.user_id == user_id)
    if account:
        transactions = (Transaction
                        .select()
                        .join(TransactionHistory)
                        .where((Transaction.sender == account) |
(Transaction.receiver == account))
                        .order_by(Transaction.timestamp.desc()))
        response = "Ваша история транзакций:\n"
        for transaction in transactions:
            history = TransactionHistory.get(
                TransactionHistory.transaction == transaction)
            direction = "отправлено" if transaction.sender == account else
"получено"
            response += f"{direction} {transaction.amount: .2f}
{transaction.currency} to/from {transaction.receiver.name if direction ==
'отправлено' else transaction.sender.name} на {transaction.timestamp}, Заметка:
{history.message}\n\n"

        bot.send_message(user_id, response)
    else:
        bot.send_message(user_id, "Ваш аккаунт не найден.")


bot.polling(none_stop=True, interval=0)
```

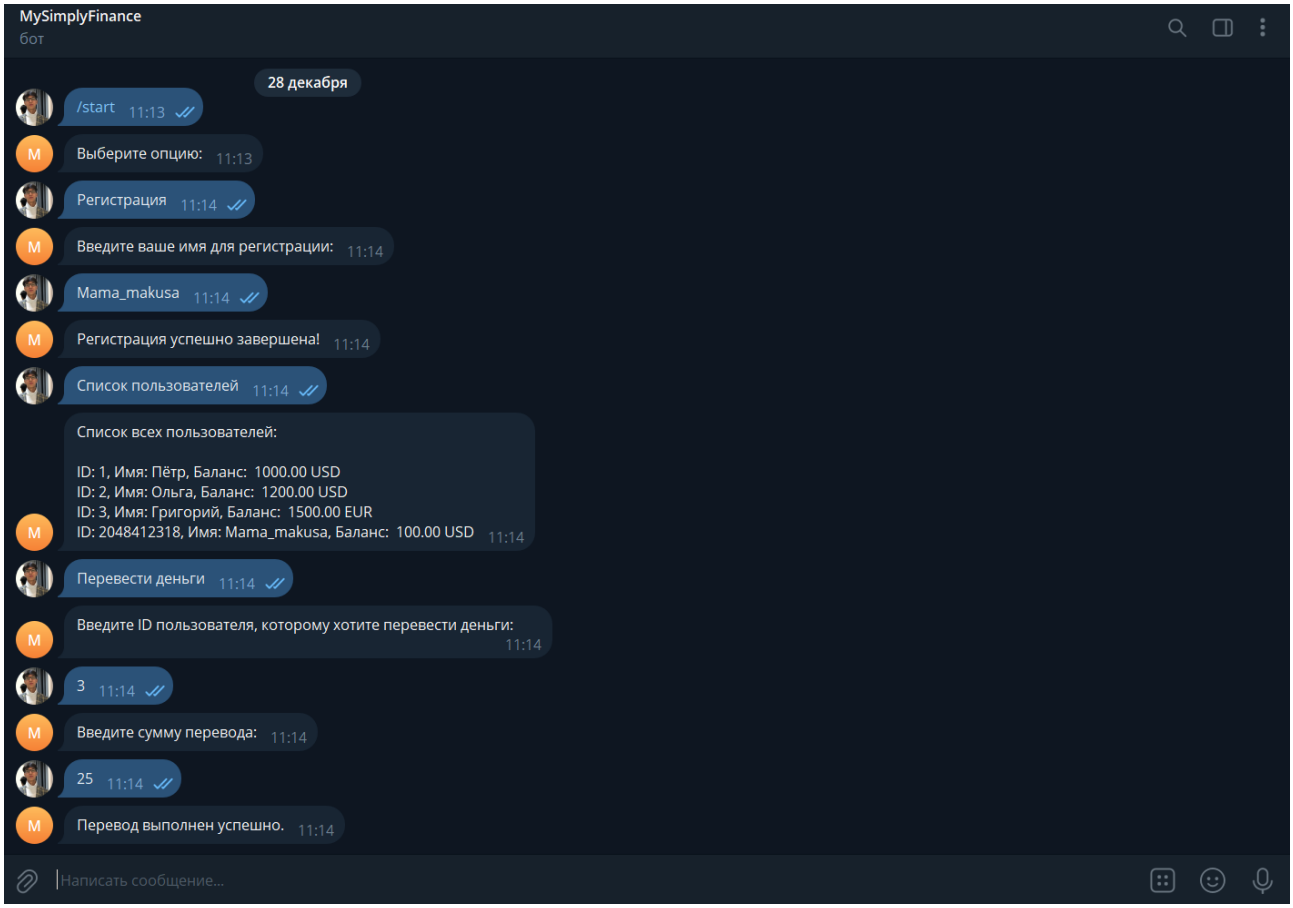
Демонстрация работы проекта.

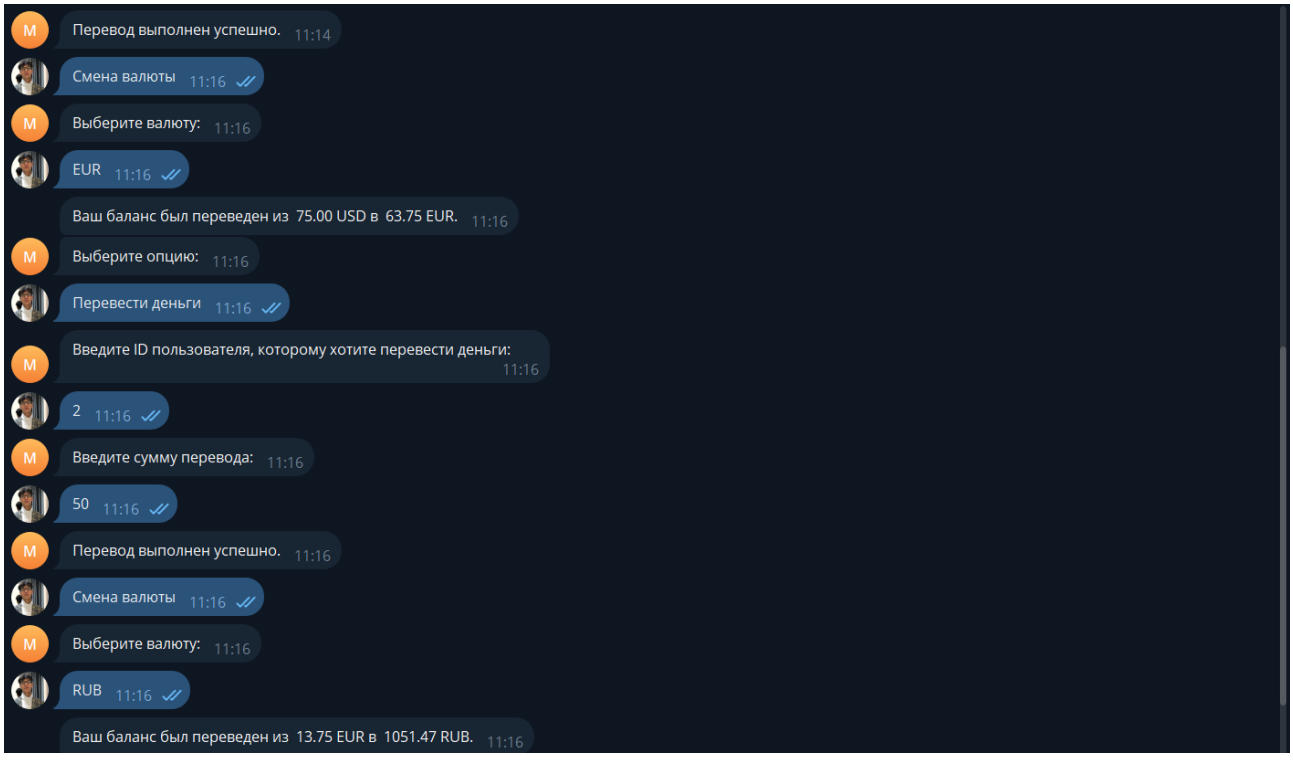
Демонстрация функций данного телеграм-бота

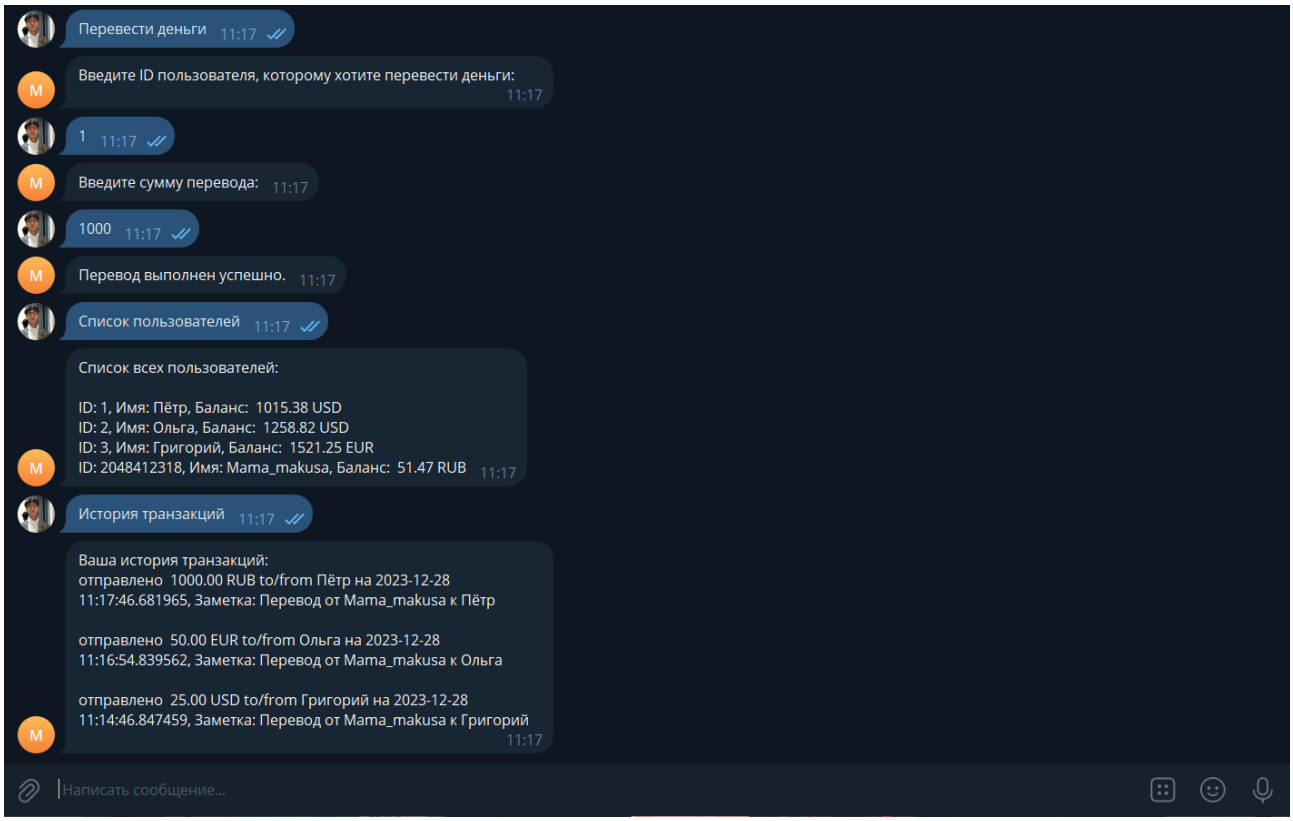
Меню:



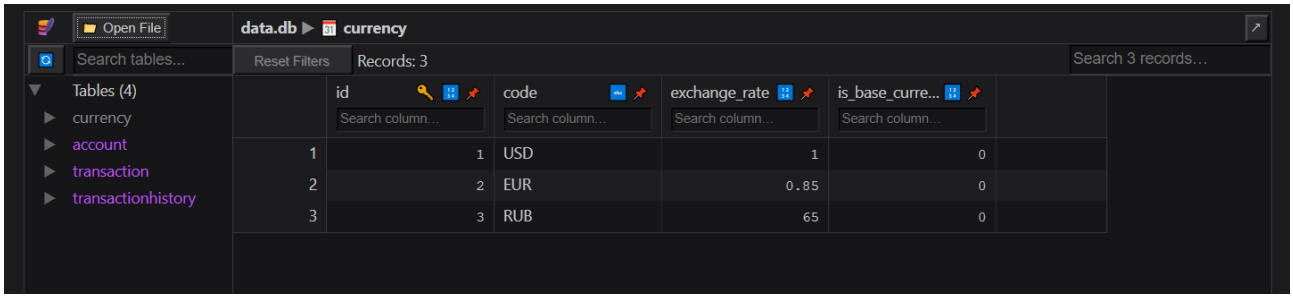
Общение с ботом:

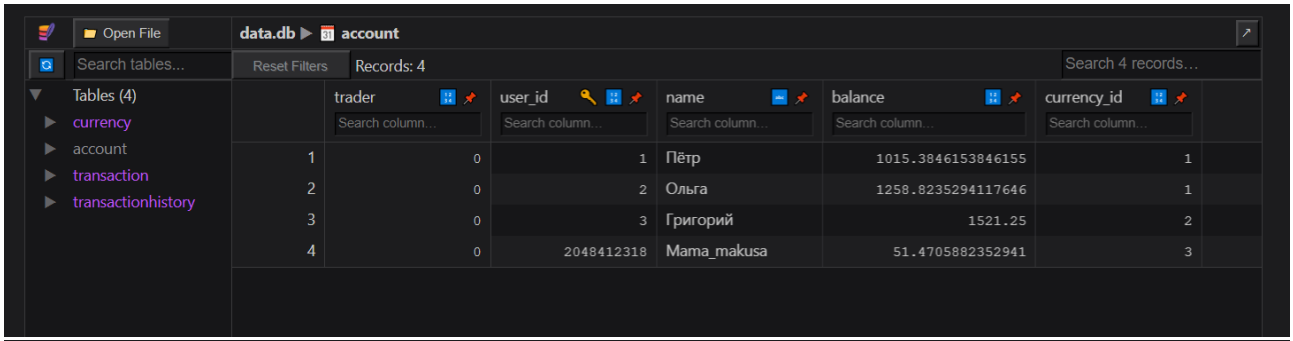


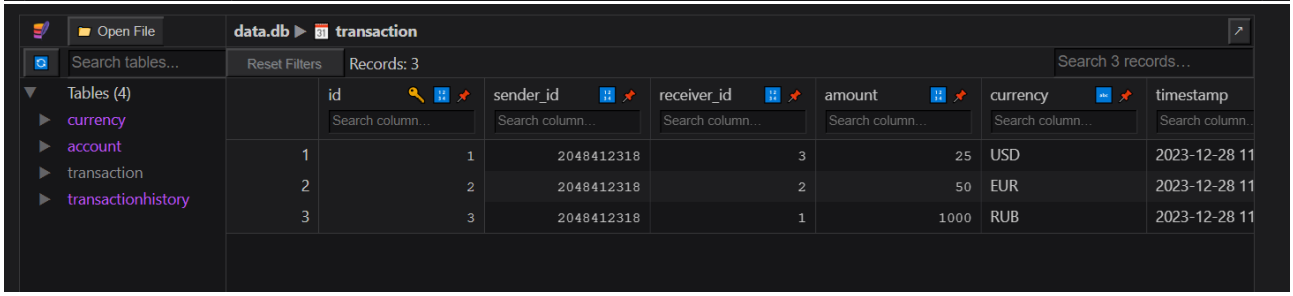


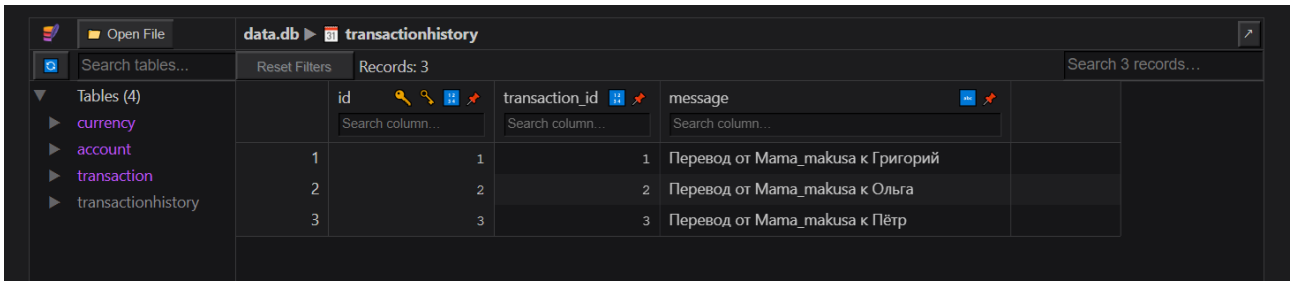


Просмотр сохраненной базы данных:








Open File
data.db ▶ transactionhistory
Search tables...
Reset Filters
Records: 3
Search 3 records...
Tables (4)
currency
account
transaction
transactionhistory
id
transaction_id
message
Search column...
Search column...
Search column...
1 1 1 Перевод от Mama_makusa к Григорий
2 2 2 Перевод от Mama_makusa к Ольга
3 3 3 Перевод от Mama_makusa к Пётр