**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: |  | Проверил: |
| студент группы ИУ5-33Б |  | преподаватель каф. ИУ5 |
| Абитов М.Р. |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | 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:

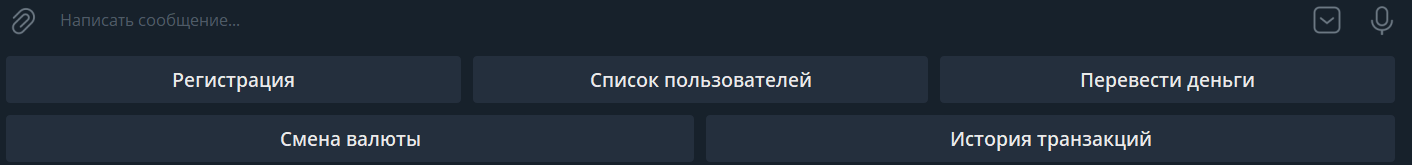
        bot.send\_message(user\_id, "Ваш аккаунт не найден.")

bot.polling(none\_stop=True, interval=0)

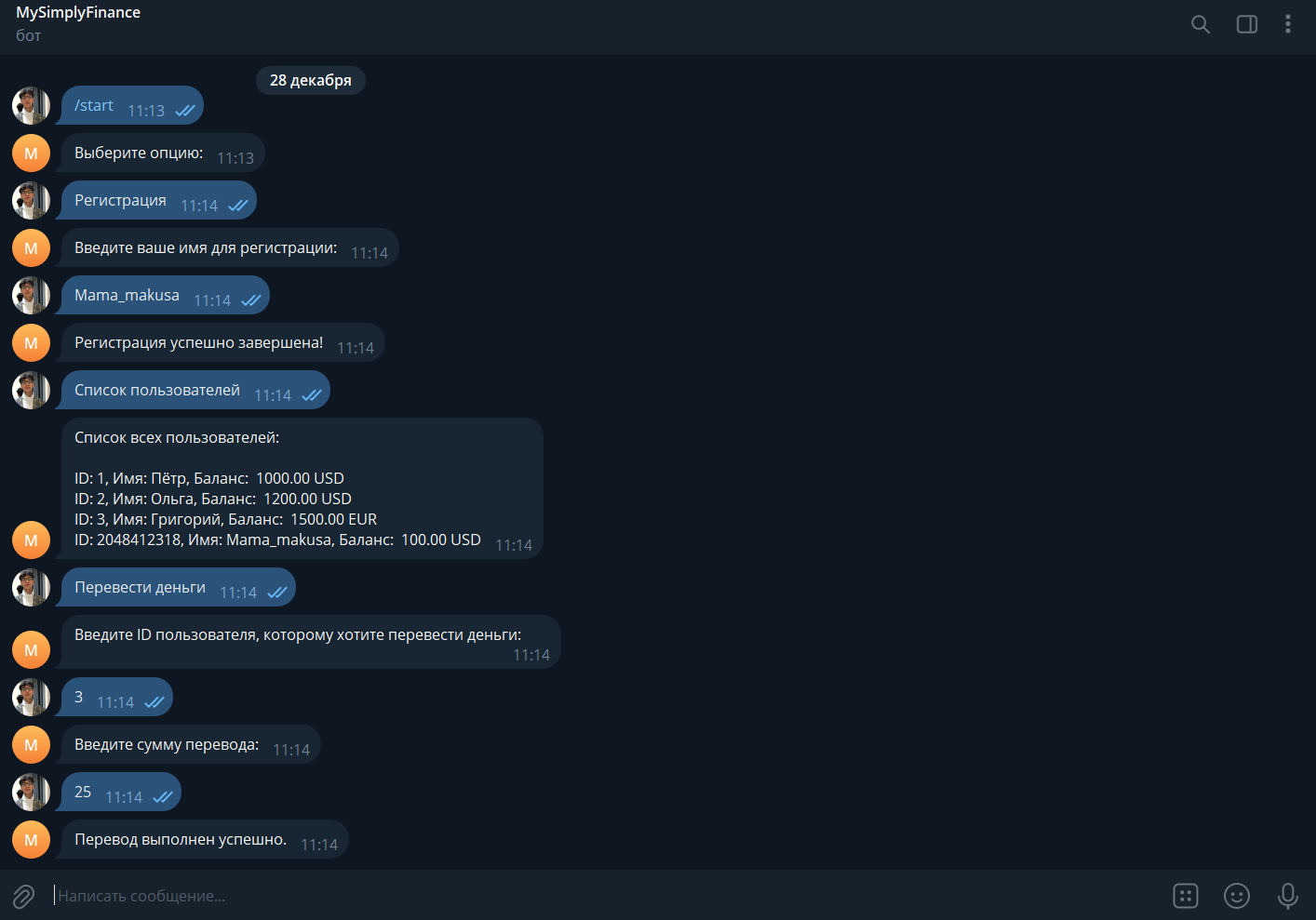
**Демонстрация работы проекта.**

Демонстрация функций данного телеграм-бота

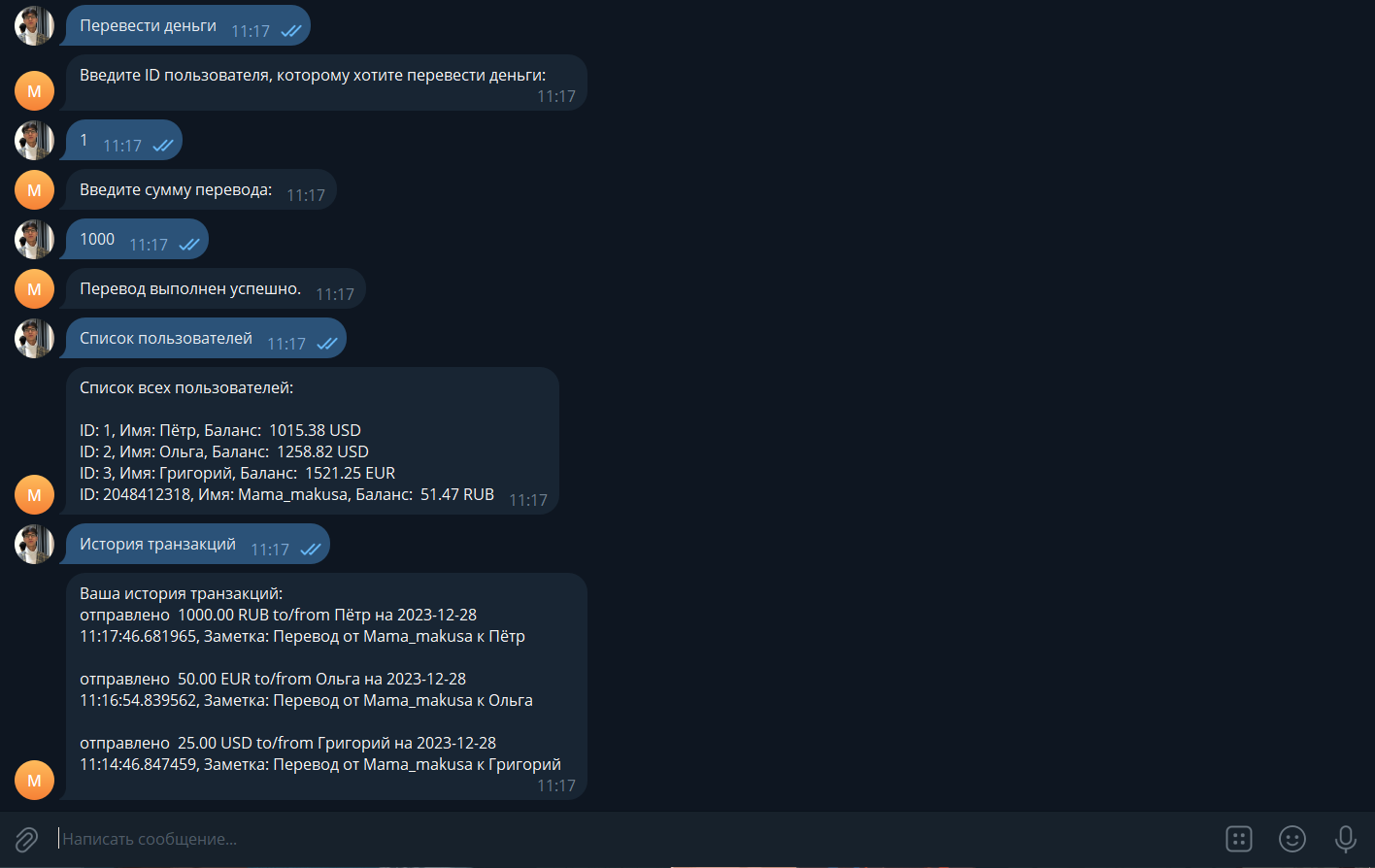
Меню:



Общение с ботом:







Просмотр сохраненной базы данных:

