*Государственное образовательное учреждение высшего профессионального образования*



***«***

***Московский***

***государственный***

***технический***

***университет***

***имени***

***Н***

***.***

***Э***

***.***

***Баумана***

***»***

***(***

***МГТУ***

***им***

***.***

***Н***

***.***

***Э***

***.***

***Баумана***

***)***

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Системы обработки информации и управления» (ИУ-5)

**ДОМАШНЕЕ ЗАДАНИЕ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**ChatFreely**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Группа ИУ5-35Б

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **17.12.2024** /**Д.Е. Мушкарин/**

(Подпись, дата) (И.О.Фамилия)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** /**Ю. Е. Гапанюк/**

(Подпись, дата) (И.О.Фамилия)

2024

**Задание:**

Разработать программу на языке Rust.

## **ChatFreely** - это анонимный чат бот, написанный на python с применением асинхронного подхода

В разработке применялись:

* aiogram [aiogram 3+ ](https://camo.githubusercontent.com/580159d8c260f71e037260cffe9667c51575a76d0011531e7128a01b5358e721/68747470733a2f2f696d672e736869656c64732e696f2f62616467652f2d332532422d726564)
* aiomysql
* pytest\_asyncio (для тестов)

Необходимый функционал:

- Механизм хранения записей всех пользователей и регулировка доступа к нему

- Соединение между любыми пользователями

- Система рейтинга, очереди соединений, профиль

- Поддержка всех типов вложений, включая стикеры, файлы

- Ответы на сообщения синхронизированы между пользователями

С целью непрерывной и надежной интеграции обновлений был реализован набор тестов для проверки основного функционала работы с базой данных.

**Текст программы:  
//main.py  
import os**

**env = os.getenv('ENV', 'default')**

**async def main():**

**await connect()**

**await create\_tables\_if\_not\_exist()**

**await start\_bot()**

**if \_\_name\_\_ == "\_\_main\_\_":**

**if env == 'default':**

**from .database import connect, create\_tables\_if\_not\_exist**

**import asyncio**

**from .bot import start\_bot**

**asyncio.run(main())**

**if env == 'test':**

**import pytest**

**pytest.main()**

**//bot.py  
from .configure import get\_key**

**from aiogram.utils.keyboard import InlineKeyboardBuilder**

**from aiogram import Router, Bot, Dispatcher, F**

**from aiogram.filters import Command**

**from aiogram.types import Message, CallbackQuery**

**from .keyboards import (**

**inline\_to\_menu\_buttons\_list,**

**inline\_banned\_buttons\_list,**

**inline\_connected\_buttons\_list,**

**inline\_dialogue\_end\_buttons\_list,**

**inline\_rate\_buttons\_list,**

**inline\_regular\_buttons\_list,**

**inline\_search\_buttons\_list**

**)**

**from .database import \***

**API\_TOKEN = get\_key()**

**bot = Bot(token=API\_TOKEN)**

**dp = Dispatcher()**

**router = Router()**

**dp.include\_router(router)**

**async def start\_bot():**

**await dp.start\_polling(bot)**

**@router.callback\_query(F.data == 'profile')**

**@router.message(Command("profile"))**

**async def profile(call):**

**if isinstance(call, CallbackQuery):**

**await call.answer('', show\_alert=False)**

**data = await fetch\_user(call.from\_user.id)**

**answer = f"""**

**Давай рассмотрим твой профиль:**

**Ваш рейтинг: {data.rating}**

**Всего диалогов: {data.total\_connections}**

**Регистрация: {data.registration}**

**"""**

**builder = InlineKeyboardBuilder()**

**builder.add(\*inline\_to\_menu\_buttons\_list)**

**markup = builder.as\_markup()**

**await bot.send\_message(chat\_id = call.from\_user.id, text= answer, reply\_markup=markup)**

**@router.message(Command("start"))**

**async def start(message: Message):**

**builder = InlineKeyboardBuilder()**

**builder.add(\*inline\_to\_menu\_buttons\_list)**

**markup = builder.as\_markup()**

**await log\_user(message.from\_user.id)**

**await message.answer(f"Привет, {message.from\_user.username}! Используй /menu, чтобы попасть в основное меню.", reply\_markup=markup)**

**@router.callback\_query(F.data == 'stop')**

**@router.message(Command("stop"))**

**async def stop\_search(call):**

**print("Stop called.")**

**if isinstance(call, CallbackQuery):**

**await call.answer('', show\_alert=False)**

**is\_searching = await is\_in\_search(call.from\_user.id)**

**if not is\_searching:**

**print(is\_searching)**

**return await menu(call)**

**await drop\_from\_search(call.from\_user.id)**

**await bot.send\_message(chat\_id = call.from\_user.id, text= f"Вы остановили поиск собеседника!")**

**await update\_status(call.from\_user.id, "normal")**

**return await menu(call)**

**@router.callback\_query(F.data == 'quit')**

**@router.message(Command("quit"))**

**async def quit\_dialogue(call):**

**if isinstance(call, CallbackQuery):**

**await call.answer('', show\_alert=False)**

**builder = InlineKeyboardBuilder()**

**builder.add(\*inline\_dialogue\_end\_buttons\_list)**

**markup = builder.as\_markup()**

**counterpart = await get\_connected\_user(call.from\_user.id)**

**if counterpart is None:**

**return await bot.send\_message(chat\_id = call.from\_user.id, text= f"Вы не находитесь в диалоге. Используйте /menu, чтобы попасть в меню, или /search, чтобы найти собеседника.", reply\_markup=markup)**

**await bot.send\_message(chat\_id = call.from\_user.id, text= f"Вы остановили диалог с вашим собеседником. Используйте /search, чтобы найти нового собеседника, или /menu для возврата в меню.", reply\_markup=markup)**

**await drop\_from\_connections(call.from\_user.id)**

**await bot.send\_message(chat\_id = counterpart, text= f"Ваш собеседник остановил диалог. Используйте /search, чтобы найти нового собеседника, или /menu для возврата в меню.", reply\_markup=markup)**

**await after\_dialogue(call.from\_user.id,counterpart)**

**builder = InlineKeyboardBuilder()**

**builder.add(\*inline\_rate\_buttons\_list)**

**builder.adjust(\*[2,1])**

**markup = builder.as\_markup()**

**await bot.send\_message(chat\_id=counterpart, text="Как вы оцените вашего последнего собеседника?",reply\_markup=markup)**

**await bot.send\_message(chat\_id=call.from\_user.id, text="Как вы оцените вашего последнего собеседника?",reply\_markup=markup)**

**@router.callback\_query(F.data == 'decrease\_rating')**

**async def decrease\_rating(call: CallbackQuery):**

**usr = await fetch\_user(call.from\_user.id)**

**if usr.last\_connected is not None:**

**await call.message.edit\_text(text="Спасибо за отзыв!")**

**await sub\_rating(usr.last\_connected, usr.telegram\_uid)**

**else:**

**await call.message.edit\_text(text="Устаревший отзыв.")**

**@router.callback\_query(F.data == 'increase\_rating')**

**async def increase\_rating(call: CallbackQuery):**

**usr = await fetch\_user(call.from\_user.id)**

**if usr.last\_connected is not None:**

**await call.message.edit\_text(text="Спасибо за отзыв!")**

**await add\_rating(usr.last\_connected, usr.telegram\_uid)**

**else:**

**await call.message.edit\_text(text="Устаревший отзыв.")**

**@router.callback\_query(F.data == 'report')**

**async def report(call: CallbackQuery):**

**usr = await fetch\_user(call.from\_user.id)**

**if usr.last\_connected is not None:**

**await call.message.edit\_text(text="Спасибо за отзыв!")**

**await add\_report(usr.last\_connected, usr.telegram\_uid)**

**else:**

**await call.message.edit\_text(text="Устаревший отзыв.")**

**@router.callback\_query(F.data == 'appeal')**

**async def report(call: CallbackQuery):**

**await call.message.edit\_text(text="Отправьте заказным письмом заявку на разблокировку с полной информацией о себе по данному адресу:")**

**await bot.send\_location(call.from\_user.id, latitude=55.766321, longitude=37.686584)**

**@router.callback\_query(F.data == 'about')**

**async def about(call: CallbackQuery):**

**await call.answer(text="", show\_alert=False)**

**ab = await bot.get\_me()**

**await bot.send\_message(chat\_id=call.from\_user.id, text=str(ab)+" located at:")**

**await bot.send\_location(call.from\_user.id, latitude=55.766321, longitude=37.686584)**

**@router.callback\_query(F.data == 'search')**

**@router.message(Command("search"))**

**async def search(call: CallbackQuery):**

**usr = await fetch\_user(call.from\_user.id)**

**if isinstance(call, CallbackQuery):**

**await call.answer('', show\_alert=False)**

**if usr.user\_status != 'normal':**

**return await menu(call)**

**await update\_status(call.from\_user.id, "search")**

**await bot.send\_dice(chat\_id = call.from\_user.id)**

**builder = InlineKeyboardBuilder()**

**builder.add(\*inline\_search\_buttons\_list)**

**# builder.adjust([1])**

**markup = builder.as\_markup()**

**await bot.send\_message(chat\_id = call.from\_user.id, text= f"Ищем для вас подходящего собеседника...", reply\_markup=markup)**

**counterpart = await get\_counterpart(call.from\_user.id)**

**if counterpart is None:**

**await add\_to\_search(usr)**

**else:**

**await drop\_from\_search(counterpart.telegram\_uid)**

**await update\_status(call.from\_user.id, "connected")**

**await update\_status(counterpart.telegram\_uid, "connected")**

**print(f"Found match for user {call.from\_user.id}(@{call.from\_user.username}), {counterpart.telegram\_uid}")**

**builder = InlineKeyboardBuilder()**

**builder.add(\*inline\_connected\_buttons\_list)**

**markup = builder.as\_markup()**

**counterpart\_answer\_rating = f"{counterpart.rating} 👍" if counterpart.rating >= 0 else f"{counterpart.rating} 👎"**

**my\_answer\_rating = f"{usr.rating} 👍" if usr.rating >= 0 else f"{usr.rating} 👎"**

**await bot.send\_message(chat\_id = call.from\_user.id, text= f"Собеседник найден, рейтинг: {counterpart\_answer\_rating} \nИспользуйте /quit, чтобы прекратить диалог.", reply\_markup=markup)**

**await bot.send\_message(chat\_id = counterpart.telegram\_uid, text= f"Собеседник найден, рейтинг: {my\_answer\_rating} \nИспользуйте /quit, чтобы прекратить диалог.", reply\_markup=markup)**

**await add\_to\_connections(call.from\_user.id,counterpart.telegram\_uid)**

**@router.callback\_query(F.data == 'menu')**

**@router.message(Command("menu"))**

**async def menu(call):**

**if isinstance(call, CallbackQuery):**

**await call.answer('', show\_alert=False)**

**await log\_user(call.from\_user.id)**

**user = await fetch\_user(call.from\_user.id)**

**status = user.user\_status**

**answer = "None"**

**if status == 'normal':**

**answer = f"Добро пожаловать в главное меню, {call.from\_user.full\_name[:50]}!"**

**buttons\_list = inline\_regular\_buttons\_list**

**elif status == 'banned':**

**answer = "Вы были заблокированы. Нажмите ниже, если хотите подать апелляцию."**

**buttons\_list = inline\_banned\_buttons\_list**

**elif status == 'search':**

**buttons\_list = inline\_search\_buttons\_list**

**answer = f"Вы в процессе поиска собеседника, {call.from\_user.full\_name[:50]}. Используйте /stop, чтобы прекратить поиск."**

**elif status == 'connected':**

**buttons\_list = inline\_connected\_buttons\_list**

**answer = f"Вы находитесь в диалоге, {call.from\_user.full\_name[:50]}. Используйте /quit, чтобы прекратить диалог."**

**keyboard\_builder = InlineKeyboardBuilder()**

**keyboard\_builder.add(\*buttons\_list)**

**keyboard\_builder.adjust(\*[1,2])**

**menu\_markup = keyboard\_builder.as\_markup()**

**await bot.send\_message(chat\_id = call.from\_user.id, text= answer, reply\_markup=menu\_markup)**

**@router.message()**

**async def send\_message(message : Message):**

**did\_reply = False if message.reply\_to\_message is None else True**

**connected = await get\_connected\_user(message.from\_user.id)**

**if connected is not None:**

**if not did\_reply:**

**reply = await message.copy\_to(chat\_id=connected)**

**else:**

**new\_reply\_id = await get\_reply\_id(message.reply\_to\_message.message\_id, message.from\_user.id)**

**# print(new\_reply\_id, "\n\n\n\n")**

**reply = await message.copy\_to(chat\_id=connected,reply\_to\_message\_id=new\_reply\_id)**

**pair = [message.message\_id, reply.message\_id]**

**await log\_message(message.from\_user.id, pair[0], pair[1])**

**await log\_message(message.from\_user.id, pair[1], pair[0])**

**else:**

**await basic(message)**

**async def basic(message : Message):**

**builder = InlineKeyboardBuilder()**

**builder.add(\*inline\_to\_menu\_buttons\_list)**

**markup = builder.as\_markup()**

**await message.answer(text="Неизвестная команда. Используйте /menu, чтобы попасть в основное меню", reply\_markup=markup)**

**//\_\_init\_\_.py  
version = "1.0.0"**

**//configure.py**

**import json**

**def read\_json\_contents(filename : str):**

**try:**

**with open(filename, "r") as file:**

**try:**

**contents = json.load(file)**

**except json.JSONDecodeError as err:**

**print(f"config.json format error in {err.doc}")**

**print(f"Error: {err.msg}")**

**print(f"At line: {err.lineno}, coloumn: {err.colno}")**

**print(f"Pos: {err.pos}")**

**file.seek(0)**

**old\_contents = file.readlines()**

**with open(f"{filename}.old", "w") as old\_file:**

**old\_file.writelines(old\_contents)**

**contents = { "Users" : []}**

**with open("config.json", "w") as file:**

**json.dump(contents, file, indent=4)**

**except FileNotFoundError:**

**print("Creating config.json...")**

**contents = { "Users" : []}**

**with open("config.json", "w") as file:**

**json.dump(contents, file, indent=4)**

**except Exception:**

**print("Unknown exception.")**

**raise Exception("The end.")**

**return contents**

**def add\_user(User):**

**contents = read\_json\_contents("config.json")**

**with open("config.json", "w") as file:**

**contents["Users"].append(User)**

**json.dump(contents, file, indent=4)**

**def list\_users():**

**contents = read\_json\_contents("config.json")**

**if contents["Users"]:**

**print("Here is the list of all the users (username@host):")**

**print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n")**

**[print(i) for i in [\_["username"]+"@"+\_["host"] for \_ in contents["Users"]]]**

**print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n")**

**else:**

**print("Database is empty\n")**

**def is\_user(username : str):**

**contents = read\_json\_contents("config.json")**

**for i in contents["Users"]:**

**if i["username"] == username:**

**return True**

**return False**

**def remove\_user(username):**

**contents = read\_json\_contents("config.json")**

**with open("config.json", "w") as file:**

**index = 0**

**for i in contents["Users"]:**

**if i["username"] == username:**

**break**

**index+=1**

**else:**

**print(f"User '{username}' not found")**

**json.dump(contents, file, indent=4)**

**return**

**del contents["Users"][index]**

**json.dump(contents, file, indent=4)**

**def update\_user(username):**

**contents = read\_json\_contents("config.json")**

**with open("config.json", "r") as file:**

**index = 0**

**for i in contents["Users"]:**

**if i["username"] == username:**

**break**

**index +=1**

**else:**

**print(f"User '{username}' not found. ")**

**return**

**while True:**

**print("What you would like to change?")**

**print("1.username")**

**print("2.host")**

**print("3.password")**

**print("4.database")**

**print("5.Save and exit edit menu")**

**a =input()**

**if not a.isdecimal():**

**print("Please, enter number from 1 to 5. Try again")**

**continue**

**a = int(a)**

**if (a < 1) or (a > 5):**

**print("Incorrect choice. Try again")**

**continue**

**if a == 1:**

**new\_username = str(input(f"Enter new username (Non-empty) for {username}: "))**

**if not new\_username:**

**print("Username can't be empty. Try again")**

**continue**

**contents["Users"][index]["username"] = new\_username**

**if a == 2:**

**new\_host = str(input(f"Enter new host (default = localhost) for {username}: "))**

**if not new\_host:**

**new\_host = "localhost"**

**contents["Users"][index]["host"] = new\_host**

**if a == 3:**

**new\_password = str(input(f"Enter new password (default = None) for {username}: "))**

**contents["Users"][index]["password"] = new\_password**

**if a == 4:**

**new\_database = str(input(f"Enter new database (default = None) for {username}: "))**

**contents["Users"][index]["database"] = new\_database**

**if a == 5:**

**break**

**with open("config.json", "w") as file:**

**json.dump(contents, file, indent=4)**

**def get\_credentials(username):**

**contents = read\_json\_contents("config.json")**

**index = 0**

**for i in contents["Users"]:**

**if i["username"] == username:**

**break**

**index +=1**

**else:**

**print(f"User {username} not found")**

**return None**

**return contents["Users"][index]**

**def get\_key():**

**contents = read\_json\_contents("config.json")**

**return contents["API"]**

**def add\_key(key : str):**

**contents = read\_json\_contents("config.json")**

**with open("config.json", "w") as file:**

**contents["API"] = key**

**json.dump(contents, file, indent=4)**

**def main():**

**while (True):**

**print(f"Welcome to the Manager control panel")**

**print("----------------------------------------")**

**print("1.Add new user credentials")**

**print("2.Remove user credentials")**

**print("3.List users")**

**print("4.Update user")**

**print("5.Add API key")**

**print("6.Exit")**

**a =input()**

**if not a.isdecimal():**

**print("Please, enter number from 1 to 5. Try again")**

**continue**

**a = int(a)**

**if (a < 1) or (a > 5):**

**print("Incorrect choice. Try again")**

**continue**

**if a == 1:**

**print("Please, enter your:")**

**User = {**

**"username" : str(input("username (Non empty): ")),**

**"host" : str(input("host (default = localhost): ")),**

**"password" : str(input("password (default = None): ")),**

**"database" : str(input("database (default = None): "))**

**}**

**if not User["username"]:**

**print("Incorrect username. Try again\n")**

**continue**

**if not User["host"]:**

**User["host"] = "localhost"**

**add\_user(User)**

**if a == 2:**

**remove\_user(str(input("username(Non empty): ")))**

**if a == 3:**

**list\_users()**

**if a == 4:**

**update\_user(str(input("username (Non empty): ")))**

**if a == 5:**

**add\_key(str(input("API key: ")))**

**if a == 6:**

**break**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

**//database.py**

**import aiomysql**

**from .configure import get\_credentials, add\_user, is\_user**

**from .user import SearchUser, User**

**import warnings**

**import random**

**import json**

**pool = None**

**async def create\_database\_async\_pool(user = "ChatFreelyAdmin"):**

**credentials = get\_credentials(user)**

**if not credentials:**

**warnings.warn("Incorrect credentials")**

**return None**

**try:**

**global pool**

**pool = await aiomysql.create\_pool(**

**user=credentials["username"],**

**db=credentials["database"],**

**password=credentials["password"],**

**host=credentials["host"],**

**minsize=1,**

**maxsize=10,**

**autocommit = True,**

**port=3306,**

**)**

**if pool is None:**

**raise Exception("Connection failed unexpectedly")**

**else:**

**print("Correct connetion acquired!")**

**except aiomysql.Error as err:**

**print("Error:", err.args)**

**return None**

**async def grace\_close():**

**if pool:**

**pool.close() # Close the pool to avoid lingering connections**

**await pool.wait\_closed() # Wait for all connections to close**

**async def connect(user : str = None):**

**if pool is not None:**

**pass**

**if user is None:**

**await create\_database\_async\_pool()**

**else:**

**print("Logging in with custom user!\n")**

**await create\_database\_async\_pool(user)**

**if pool is None:**

**raise BaseException("Could not resolve mysql database connection. Maybe check credentials/start db.")**

**async def create\_tables\_if\_not\_exist():**

**warnings.filterwarnings("ignore", message="Table '.\*' already exists")**

**async with pool.acquire() as conn:**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**CREATE TABLE IF NOT EXISTS users (**

**telegram\_uid BIGINT PRIMARY KEY NOT NULL,**

**user\_status ENUM('normal', 'connected', 'search', 'banned') DEFAULT 'normal',**

**INDEX (user\_status),**

**rating INT DEFAULT 0,**

**registration TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,**

**total\_connections INT DEFAULT 0,**

**last\_update TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,**

**last\_connected BIGINT NULL DEFAULT NULL,**

**reports INT DEFAULT 0**

**);**

**""")**

**await cursor.execute(**

**"""**

**CREATE TABLE IF NOT EXISTS search (**

**telegram\_uid BIGINT PRIMARY KEY NOT NULL,**

**rating INT DEFAULT 0,**

**FOREIGN KEY (telegram\_uid) REFERENCES users(telegram\_uid) ON DELETE CASCADE**

**);**

**""")**

**await cursor.execute(**

**"""**

**CREATE TABLE IF NOT EXISTS connections (**

**telegram\_uid\_1 BIGINT PRIMARY KEY NOT NULL,**

**FOREIGN KEY (telegram\_uid\_1) REFERENCES users(telegram\_uid) ON DELETE CASCADE,**

**telegram\_uid\_2 BIGINT NOT NULL,**

**FOREIGN KEY (telegram\_uid\_2) REFERENCES users(telegram\_uid) ON DELETE CASCADE,**

**UNIQUE INDEX T\_UID\_2(telegram\_uid\_2),**

**messages\_table JSON DEFAULT '{}'**

**);**

**""")**

**async def drop\_tables(): #ОСТОРОЖНО!!**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**try:**

**warnings.filterwarnings("ignore", message="Unknown table '.\*'")**

**await cursor.execute("DROP TABLE IF EXISTS connections;")**

**await cursor.execute("DROP TABLE IF EXISTS search;")**

**await cursor.execute("DROP TABLE IF EXISTS users;")**

**except aiomysql.OperationalError as err:**

**warnings.warn(f"Err: {err.args}")**

**await conn.rollback()**

**return False**

**await conn.commit()**

**return True**

**async def log\_user(telegram\_uid):**

**# print(f"Message {context} logged.")**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**INSERT INTO users**

**(**

**telegram\_uid**

**)**

**VALUES (%s)**

**ON DUPLICATE KEY UPDATE**

**last\_update = CURRENT\_TIMESTAMP**

**""", (telegram\_uid,))**

**await conn.commit()**

**async def drop\_user(telegram\_uid):**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**DELETE FROM users WHERE telegram\_uid = %s**

**""", (telegram\_uid,))**

**async def has\_data():**

**res = {"users" : 0, "connections" : 0, "search" : 0}**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**SELECT \* FROM users;**

**""")**

**if cursor.rowcount > 0:**

**res["users"] = 1**

**await cursor.execute(**

**"""**

**SELECT \* FROM connections;**

**""")**

**if cursor.rowcount > 0:**

**res["connections"] = 1**

**await cursor.execute(**

**"""**

**SELECT \* FROM search;**

**""")**

**if cursor.rowcount > 0:**

**res["search"] = 1**

**await conn.commit()**

**return res**

**async def add\_to\_search(user : User):**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**REPLACE INTO search**

**(**

**telegram\_uid,**

**rating**

**)**

**VALUES (%s,%s);**

**""", (user.telegram\_uid, user.rating))**

**await conn.commit()**

**print("User added!")**

**async def drop\_from\_search(telegram\_uid):**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**DELETE FROM search**

**WHERE telegram\_uid = %s**

**""", (telegram\_uid, ))**

**await conn.commit()**

**async def add\_to\_connections(recipient, counterpart):**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**REPLACE INTO connections**

**(**

**telegram\_uid\_1,**

**telegram\_uid\_2**

**)**

**VALUES (%s,%s)**

**;**

**""", (recipient, counterpart))**

**await conn.commit()**

**async def drop\_from\_connections(telegram\_uid):**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**DELETE FROM connections**

**WHERE telegram\_uid\_1 = %s**

**OR telegram\_uid\_2 = %s**

**""", (telegram\_uid, telegram\_uid))**

**await conn.commit()**

**async def update\_status(uid : str, status : str):**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**UPDATE users**

**SET user\_status = %s**

**WHERE telegram\_uid = %s**

**""", (status,uid))**

**await conn.commit()**

**async def after\_dialogue(uid : str, uid2 : str):**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**UPDATE users**

**SET user\_status = 'normal', total\_connections = total\_connections + 1, last\_connected = %s**

**WHERE telegram\_uid = %s;**

**""", (uid, uid2))**

**await cursor.execute(**

**"""**

**UPDATE users**

**SET user\_status = 'normal', total\_connections = total\_connections + 1, last\_connected = %s**

**WHERE telegram\_uid = %s;**

**""", (uid2, uid))**

**await conn.commit()**

**async def fetch\_user(user\_id):**

**async with pool.acquire() as conn:**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**SELECT \* FROM users WHERE telegram\_uid = %s**

**""", (user\_id,))**

**if cursor.rowcount==0:**

**print(f"An attempt to fetch unknown user: {user\_id}")**

**return None**

**data = await cursor.fetchone()**

**return User(data)**

**async def get\_connected\_user(telegram\_uid : str):**

**async with pool.acquire() as conn:**

**async with conn.cursor() as cursor:**

**await cursor.execute(**

**"""**

**SELECT \* FROM connections WHERE telegram\_uid\_1 = %s OR telegram\_uid\_2 = %s;**

**""", (telegram\_uid, telegram\_uid))**

**if cursor.rowcount > 0:**

**data = await cursor.fetchone()**

**if data[0]==telegram\_uid:**

**return data[1]**

**return data[0]**

**else:**

**print(f"No user {telegram\_uid} in connections.")**

**return None**

**async def is\_in\_search(telegram\_uid):**

**async with pool.acquire() as conn:**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**SELECT \* FROM search**

**WHERE telegram\_uid = %s**

**""", (telegram\_uid,))**

**# await conn.commit()**

**if cursor.rowcount > 0:**

**res = True**

**else:**

**res = False**

**# await conn.commit()**

**return res**

**async def get\_counterpart(telegram\_uid : str):**

**async with pool.acquire() as conn:**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**user = await fetch\_user(telegram\_uid)**

**user : User**

**await cursor.execute(**

**"""**

**SELECT \* FROM search**

**WHERE telegram\_uid != %s**

**ORDER BY ABS(rating-%s);**

**""", (user.telegram\_uid, user.rating ))**

**if cursor.rowcount > 0:**

**data = await cursor.fetchone()**

**return SearchUser(data)**

**else:**

**print(f"Match for {user.telegram\_uid, user.rating } Not Found.")**

**return None**

**async def add\_rating(telegram\_uid, voter\_uid):**

**async with pool.acquire() as conn:**

**async with conn.cursor() as cursor:**

**cursor : aiomysql.Cursor**

**await cursor.execute("""**

**UPDATE users**

**SET rating = rating + 1**

**WHERE telegram\_uid = %s;**

**""", (telegram\_uid,))**

**await cursor.execute("""**

**UPDATE users**

**SET last\_connected = NULL**

**WHERE telegram\_uid = %s;**

**""", (voter\_uid,))**

**await conn.commit()**

**async def sub\_rating(telegram\_uid, voter\_uid):**

**async with pool.acquire() as conn:**

**async with conn.cursor() as cursor:**

**cursor : aiomysql.Cursor**

**await cursor.execute("""**

**UPDATE users**

**SET rating = rating - 1**

**WHERE telegram\_uid = %s;**

**""", (telegram\_uid,))**

**await cursor.execute("""**

**UPDATE users**

**SET last\_connected = NULL**

**WHERE telegram\_uid = %s;**

**""", (voter\_uid,))**

**await conn.commit()**

**async def add\_report(telegram\_uid, voter\_uid):**

**async with pool.acquire() as conn:**

**async with conn.cursor() as cursor:**

**cursor : aiomysql.Cursor**

**await cursor.execute("""**

**UPDATE users**

**SET reports = reports + 1**

**WHERE telegram\_uid = %s;**

**""", (telegram\_uid,))**

**await cursor.execute("""**

**UPDATE users**

**SET last\_connected = NULL**

**WHERE telegram\_uid = %s;**

**""", (voter\_uid,))**

**await conn.commit()**

**async def log\_message(sender\_uid, from\_message\_id, bot\_message\_id):**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**SELECT messages\_table FROM connections WHERE telegram\_uid\_1 = %s OR telegram\_uid\_2 = %s;**

**""",(sender\_uid, sender\_uid))**

**fetch = await cursor.fetchone()**

**obj = json.loads(fetch[0])**

**obj[from\_message\_id] = bot\_message\_id**

**if len(obj) > 20:**

**res = {}**

**counter = 0**

**for key, value in obj.items():**

**counter +=1**

**if counter > 10:**

**res[key] = value**

**obj = res**

**load = json.dumps(obj)**

**await cursor.execute(**

**"""**

**UPDATE connections SET messages\_table = %s WHERE telegram\_uid\_1 = %s OR telegram\_uid\_2 = %s;**

**""",(load, sender\_uid, sender\_uid))**

**await conn.commit()**

**async def get\_reply\_id(message\_id,from\_message\_id):**

**async with pool.acquire() as conn:**

**conn : aiomysql.Connection**

**async with conn.cursor() as cursor:**

**cursor: aiomysql.Cursor**

**await cursor.execute(**

**"""**

**SELECT messages\_table FROM connections WHERE telegram\_uid\_1 = %s OR telegram\_uid\_2 = %s;**

**""",(from\_message\_id, from\_message\_id))**

**fetch = await cursor.fetchone()**

**# if fetch is None:**

**# print(f"Message reply not found for users {from\_message\_id}, {bot\_message\_id}")**

**res = json.loads(fetch[0])**

**res : dict**

**reply\_id = res.get(str(message\_id))**

**return reply\_id**

**async def get\_two\_unique():**

**unique = []**

**while True:**

**again = False**

**unique = [random.randint(0, 1000000) for i in range(2)]**

**for item in unique:**

**if unique.count(item) > 1:**

**again = True**

**break**

**if not again:**

**break**

**return unique**

**async def prepare\_test\_env():**

**if not is\_user("test\_user"):**

**add\_user({"username" : "test\_user", "password" : "test\_password", "host" : "localhost", "database" : "test\_db"})**

**await connect("test\_user")**

**//keyboards.py**

**from aiogram.types import InlineKeyboardButton**

**inline\_to\_menu\_buttons\_list = [**

**InlineKeyboardButton(text="Перейти в меню", callback\_data="menu")**

**]**

**inline\_dialogue\_end\_buttons\_list = [**

**InlineKeyboardButton(text="Перейти в меню", callback\_data="menu"),**

**InlineKeyboardButton(text="Найти собеседника", callback\_data="search")**

**]**

**inline\_search\_buttons\_list = [**

**InlineKeyboardButton(text="Прекратить поиск", callback\_data="stop")**

**]**

**inline\_connected\_buttons\_list = [**

**InlineKeyboardButton(text="Прекратить диалог", callback\_data="quit")**

**]**

**inline\_regular\_buttons\_list = [**

**InlineKeyboardButton(text="Найти собеседника", callback\_data="search"),**

**InlineKeyboardButton(text="Мой профиль", callback\_data="profile"),**

**InlineKeyboardButton(text="О боте", callback\_data="about")**

**]**

**inline\_banned\_buttons\_list = [**

**InlineKeyboardButton(text="Подать апелляцию", callback\_data="appeal")**

**]**

**inline\_rate\_buttons\_list = [**

**InlineKeyboardButton(text="👍", callback\_data="increase\_rating"),**

**InlineKeyboardButton(text="👎", callback\_data="decrease\_rating"),**

**InlineKeyboardButton(text="Пожаловаться", callback\_data="report")   
]**

**//user.py**

**class BaseUser():**

**def \_\_init\_\_(self, data):**

**self.\_telegram\_uid = data[0]**

**@property**

**def telegram\_uid(self):**

**return self.\_telegram\_uid**

**@telegram\_uid.setter**

**def telegram\_uid(self, uid):**

**self.\_telegram\_uid = uid**

**class SearchUser(BaseUser):**

**def \_\_init\_\_(self, data):**

**self.\_telegram\_uid = data[0]**

**self.\_rating = data[1]**

**@property**

**def rating(self):**

**return self.\_rating**

**@rating.setter**

**def rating(self, rating):**

**self.\_rating = rating**

**class User(SearchUser):**

**def \_\_init\_\_(self, data):**

**self.\_telegram\_uid = data[0]**

**self.\_user\_status = data[1]**

**self.\_rating = data[2]**

**self.\_registration = data[3]**

**self.\_total\_connections = data[4]**

**self.\_last\_update = data[5]**

**self.\_last\_connected = data[6]**

**self.\_reports = data[7]**

**@property**

**def connected\_uid(self):**

**return self.\_connected\_uid**

**@connected\_uid.setter**

**def connected\_uid(self, uid):**

**self.\_connected\_uid = uid**

**@property**

**def registration(self):**

**return self.\_registration**

**@registration.setter**

**def registration(self, registration):**

**self.\_registration = registration**

**@property**

**def user\_status(self):**

**return self.\_user\_status**

**@user\_status.setter**

**def user\_status(self, status):**

**self.\_user\_status = status**

**@property**

**def last\_update(self):**

**return self.\_last\_update**

**@last\_update.setter**

**def last\_update(self, update):**

**self.\_last\_update = update**

**@property**

**def total\_connections(self):**

**return self.\_total\_connections**

**@total\_connections.setter**

**def total\_connections(self, total\_connections):**

**self.\_total\_connections = total\_connections**

**@property**

**def last\_connected(self):**

**return self.\_last\_connected**

**@last\_connected.setter**

**def last\_connected(self, last\_connected):**

**self.\_last\_connected = last\_connected**

**@property**

**def reports(self):**

**return self.\_reports**

**@reports.setter**

**def reports(self, reports):**

**self.\_reports= reports**

**class ConnectedUser(BaseUser):**

**def \_\_init\_\_(self, data):**

**super().\_\_init\_\_(data[0])**

**self.\_telegram\_uid = data[1]**

**@property**

**def telegram\_uid\_2(self):**

**return self.\_telegram\_uid**

**@telegram\_uid\_2.setter**

**def telegram\_uid\_2(self, uid):**

**self.\_telegram\_uid = uid**

**//conftest.py**

**import pytest\_asyncio**

**from ChatFreelyBot.database import grace\_close, drop\_tables, create\_tables\_if\_not\_exist, connect, prepare\_test\_env**

**@pytest\_asyncio.fixture(loop\_scope="function")**

**async def module\_setup\_teardown():**

**await connect("test\_user")**

**await drop\_tables()**

**await create\_tables\_if\_not\_exist()**

**await prepare\_test\_env()**

**yield True**

**await drop\_tables()**

**await grace\_close()**

**//test\_module\_1.py**

**# test\_module\_1.py**

**import pytest**

**import warnings**

**from ChatFreelyBot.database import (**

**log\_user, fetch\_user,**

**drop\_user, get\_two\_unique,**

**has\_data, add\_to\_search,**

**add\_to\_connections, drop\_tables,**

**drop\_from\_search, get\_counterpart)**

**@pytest.mark.usefixtures("module\_setup\_teardown")**

**class TestClass:**

**@pytest.mark.asyncio**

**async def test\_add\_drop\_user(module\_setup\_teardown): # тест добавления и удаления пользователя**

**test\_uids = await get\_two\_unique()**

**test\_uid = test\_uids[0]**

**await log\_user(test\_uid)**

**usr = await fetch\_user(test\_uid)**

**assert usr.telegram\_uid == test\_uid \**

**and usr.reports == 0 \**

**and usr.total\_connections == 0 \**

**and usr.rating == 0 \**

**and usr.user\_status == 'normal' # пользователь создается нормальным**

**await drop\_user(test\_uid)**

**usr = await fetch\_user(test\_uid)**

**assert usr is None # пользователь действительно удаляется**

**@pytest.mark.asyncio # продвинутый тест множественного добавления и удаления**

**async def test\_add\_drop\_user\_two(module\_setup\_teardown):**

**test\_uids = await get\_two\_unique()**

**test\_uid = test\_uids[0]**

**await log\_user(test\_uid)**

**await log\_user(test\_uid)**

**usr = await fetch\_user(test\_uid)**

**assert usr is not None # идентичное создание**

**usr1 = await fetch\_user(test\_uid)**

**usr2 = await fetch\_user(test\_uid)**

**assert (usr1.registration == usr2.registration) # идентичное получение**

**await drop\_user(test\_uid)**

**usr = await fetch\_user(test\_uid)**

**assert usr is None # нет двух записей**

**@pytest.mark.asyncio**

**async def test\_is\_empty(module\_setup\_teardown):**

**res = await has\_data()**

**for value in res.values():**

**assert not value**

**test\_uids = await get\_two\_unique()**

**for uid in test\_uids:**

**await log\_user(uid)**

**await add\_to\_search(await fetch\_user(uid))**

**await add\_to\_connections(test\_uids[0], test\_uids[1])**

**res = await has\_data()**

**for value in res.values():**

**assert value**

**@pytest.mark.asyncio**

**async def test\_drop\_tables(module\_setup\_teardown): # тест очистки таблиц**

**warnings.filterwarnings(message="Dropping tables from the empty database", action='ignore')**

**code = await drop\_tables()**

**assert code**

**@pytest.mark.asyncio**

**async def test\_add\_drop\_user\_multiple(module\_setup\_teardown): # нагрузочный тест**

**for entry in range(125):**

**await log\_user(entry)**

**await drop\_user(entry)**

**res = await has\_data()**

**for value in res.values():**

**assert not value**

**@pytest.mark.asyncio**

**async def test\_add\_drop\_search(module\_setup\_teardown): # тест корретного соединения**

**test\_uids = await get\_two\_unique()**

**for uid in test\_uids:**

**await log\_user(uid)**

**usr = await fetch\_user(uid)**

**await add\_to\_search(usr)**

**uid1\_counterpart = await get\_counterpart(test\_uids[0])**

**assert (uid1\_counterpart.telegram\_uid == test\_uids[1])**

**uid2\_counterpart = await get\_counterpart(test\_uids[1])**

**assert (uid2\_counterpart.telegram\_uid == test\_uids[0]) # верное соединение**

**for uid in test\_uids:**

**await drop\_from\_search(uid)**

**await drop\_user(uid) # убираем из бд**

**res = await has\_data()**

**for value in res.values():**

**assert not value # очищение работает верно**