**ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ**

****

**ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ГОРОДА МОСКВЫ**

**«ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ ИМЕНИ П.А. ОВЧИННИКОВА»**

**Специальность:**

09.02.07 Информационные системы и программирование

**КУРСОВАЯ РАБОТА**

**МДК.01.01. Разработка программных модулей**

тема: «Разработка мессенджера на языке программирования Python»

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студент: | Комаеда Надежда Юлиевна | | | | |  | | |
|  | |  | **Ф.И.О** | | | **подпись** | | |
| Группа №: | | 2ИС |  | |  |  | |  |
|  | |  | |  | |  | |  |
| Руководитель: | | Олейник Владимир Иванович | | | |  | |  |
|  | |  | **Ф.И.О** | | | **подпись** | | |
|  | |  | | | |  |  |  |
| Оценка: |  | | Дата: | | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2023 г. | | | |
|  | |  | | | |  | |  |

**Москва**

**2023**

**Оглавление**

Введение 1

1. Разработка мессенджера 2

1.1. Анализ существующих решений 2

1.2. Суть приложения 2

2. Составление требований к программе 3

2.1. Введение 3

2.2. Требования к программному продукту. 3

3. Проектирование архитектуры 4

4. Проектирование интерфейсов 5

4.1. Клиент 5

4.2. Сервер 18

5. Кодирование 18

5.1. Создание сервера и подключение к серверу 18

5.2. Обмен данными 22

5.3. Вход в аккаунт 25

5.4. Регистрация пользователей 31

5.6. Основные виджеты и переменные главного окна мессенджера 35

5.7. Настройки аккаунта 39

5.8. Контакты 46

5.9. Сообщения 57

Заключение 64

Источники 65

Введение

*«В начале было Слово…»*

*От Иоанна 1:1*

Общение запустило двигатель прогресса: объединило первые племена, основало первые цивилизации, совершило первые открытия, и, в конце концов, привело нас к сегодняшнему дню. Общение – это неотъемлемая часть человеческой сущности. По мере развития общества появлялись новые и новые средства коммуникации: сначала, как предполагают ученые, это были жесты и звуки, потом появилась речь, затем письмо. В последние сто лет технологии развиваются экспоненциально, и вместе с новыми открытиями появились и новые возможности.

Возможность обмениваться информацией мгновенно стала неотъемлемой частью жизни современного человека, в том числе и меня, поэтому я решила выбрать написание мессенджера на языке программирования Python в качестве темы для моей курсовой работы.

Мессенджеры позволяют лучше планировать и координировать действия на работе, помогают получать важные новости, оставаться на связи с семьей и друзьями в реальном времени даже на большом расстоянии.

Язык программирования Python предлагает широкий набор библиотек, которые облегчают процесс разработки. К тому же, большинство команд интуитивно понятны, ведь многие из них это аббревиатуры, словосочетания или слова на английском языке.

1. Разработка мессенджера

# 1.1. Анализ существующих решений

1. Для работы с языком программирования Python существует множество ПО. Для разработки мессенджера я буду использовать PyCharm, так как у меня есть опыт работы с ним, он интуитивен в использовании и имеет широкий функционал.
2. Для хранения информации понадобится база данных. Для этой цели я буду использовать Microsoft Excel. Для работы с данным ПО я буду пользоваться библиотекой openpyxl.
3. Для обеспечения коммуникации между клиентом и сервером я буду использовать модуль socket. Чтобы сервер мог работать с несколькими клиентами сразу, понадобится модуль threading, обеспечивающий многозадачность.
4. Для создания окон я буду использовать модуль tkinter из встроенной библиотеки Python.
5. Для облегчения процесса разработки я буду использовать GitHub.

# 1.2. Суть приложения

Мессенджер должен обеспечивать следующие возможности:

1. Регистрация, входа в аккаунт.
2. Изменение фотографии профиля из предложенного набора картинок, изменение юзернейма и пароля, выход из аккаунта.
3. Поиск пользователей по юзернейму, просмотр контактов.
4. Отправление сообщений, обновление истории в реальном времени, очистка истории переписки.
5. Закрытие программы.

Сервер должен позволять работать с несколькими клиентами сразу, изменять и читать базу данных по запросу клиента, печатать отчеты в консоль.

1. Составление требований к программе

# Введение

**Наименование программного продукта**

Наименование: checkmate

**Основание для разработки**

Выполнение курсовой работы, углубление знаний в языке программирования Python

**Назначение разработки**

Цель проекта – разработка мессенджера, который представляет собой небольшое экранное приложение. Он позволит пользователям обмениваться сообщениями в режиме реального времени. Проект планируется совершенствовать в будущем.

**Область применения**

Предназначен для широкой аудитории пользователей всех возрастных групп; предполагается, что он будет использоваться для личных целей.

# Требования к программному продукту.

Сервер должен обеспечивать выполнение перечисленных ниже функций:

1. Запуск сервера
2. Печать отчета о каждом запросе с клиента в консоль
3. Прием и отправка запросов, обеспечение целостности данных
4. Выполнение определенных действий в соответствии с полученными запросами
5. Многозадачность: работа с каждым подключенным клиентом по отдельности
6. Изменение и чтение базы данных
7. Запись историй сообщений пользователей в текстовые файлы
8. Обработка ошибок

Клиент должен обеспечивать выполнение перечисленных ниже функций:

1. Подключение к серверу
2. Прием и отправление запросов, обеспечение целостности данных
3. Выполнение определенных действий в соответствии с полученными запросами
4. Вход в аккаунт, выход из аккаунта; автоматический вход в аккаунт при запуске приложения, если выход выполнен не был
5. Регистрация пользователей
6. Изменение фотографии профиля
7. Изменение юзернейма и пароля
8. Поиск пользователей
9. Просмотр контактов
10. Отправление сообщений другим пользователям
11. Обновление истории переписки в реальном времени
12. Очистка истории сообщений
13. Закрытие программы
14. Обработка ошибок
15. Проектирование архитектуры

Для разработки мессенджера была выбрана клиент-серверная архитектура. В данной модели есть два основных компонента: клиент, который работает на устройстве пользователя, и сервер, который обрабатывает запросы от клиентов.

Данная архитектура обеспечивает возможность взаимодействия нескольких пользователей одновременно.

Клиент отвечают только за пользовательский интерфейс и взаимодействие с пользователем. Сервер же обрабатывает принятые от клиента запросы и хранит данные.

**Разделение компонентов**

Клиент:

1. Отвечает за пользовательский интерфейс и взаимодействие с пользователем  
2. Отправляет запросы на сервер и получает ответы с сервера

Сервер:

1. Обрабатывает запросы, полученные от клиента
2. Читает и изменяет базу данных

База данных:

Хранит истории переписок и информацию о пользователях.

4. Проектирование интерфейсов

# 4.1. Клиент

При запуске программы в первый раз появляется окно входа в аккаунт (Рис. 1.1). Данное окно содержит поля ввода «логин» и «пароль», куда предлагается ввести соответствующие данные если пользователь уже зарегистрировался. В целях безопасности вместо вводимых символов в поле ввода «пароль» пользователь видит звездочки.

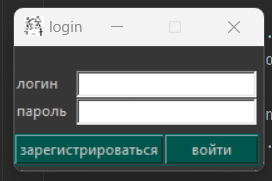
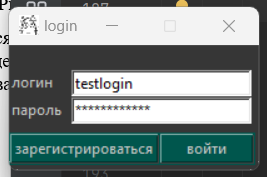
 

Рис. 1.1 «Окно входа в аккаунт»

Если пользователь ввел в поле ввода текст, содержащий символ «:», и нажал на кнопку «войти», на экран выведется сообщение «недопустимые символы (:)» (Рис 1.2). Это сделано затем, чтобы предотвратить ошибку: сервер принимает запрос от клиента и разделяет полученное сообщение по знаку «:» (login:password).

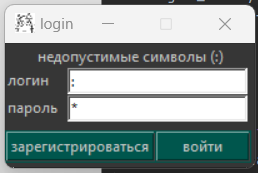


Рис 1.2 «недопустимые символы (:)»

Если пользователь ввел в поле логина «login» или в поле пароля «password» и нажал на кнопку «войти», на экран выведется сообщение «не-а» (Рис 1.3). Это сделано затем, чтобы исключить ошибку при чтении базы данных, так как первая строка таблицы содержит колонки именно с такими названиями.

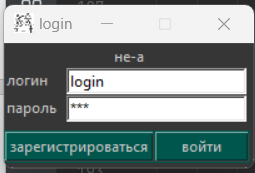
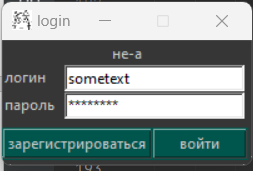
 

Рис 1.3 «не-а» (на втором скриншоте в поле «пароль» введено «password»)  
  
 Если введенные пользователем логин и пароль не совпадают с информацией в базе данных, после нажатия на кнопку «войти», на экран выведется сообщение «>\_< некорректные данные» (Рис. 1.4)

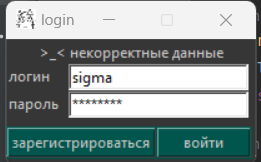


Рис 1.4 «некорректные данные»  
  
 Если пользователь еще не зарегистрирован, то ему предлагается создать аккаунт. Для этого нужно нажать на кнопку «зарегистрироваться», после чего появляется окно регистрации (Рис 2.1) с полями «юзернейм» (то, что будет отображаться в профиле), «логин» (с помощью него будет производиться вход в аккаунт), «пароль» и «повторите пароль» (нужно для уверенности в том, что пользователь ввел желаемый пароль корректно). Также слева есть кнопка «🡨», по нажатию на которую пользователя возвращает на окно входа в аккаунт.

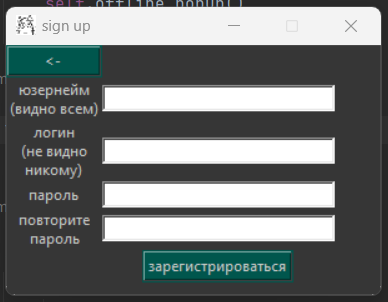


Рис. 2.1 «Окно регистрации»

Если хотя бы одно из полей пустое, на экран выведется сообщение «заполните все поля» после нажатия на кнопку «зарегистрироваться». Если пользователь ввел хотя бы в одно из полей текст, содержащий символ «:», после нажатия на кнопку «зарегистрироваться» на экран выведется сообщение «недопустимые символы (:)». (Рис. 2.2., Рис. 2.3)

Если пользователь ввел в поле «юзернейм» текст, содержащий более 10 символов, после нажатия на кнопку «зарегистрироваться» на экран выведется сообщение «слишком длинный юзернейм», и содержимое соответствующего поля будет обрезано до 10 символов. Это нужно для того, чтобы юзернейм правильно отображался. (Рис. 2.4.)

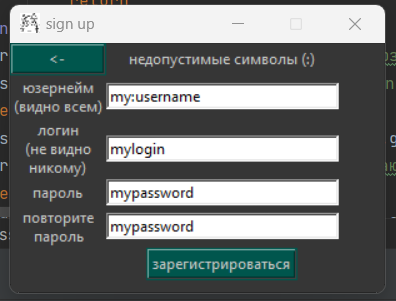
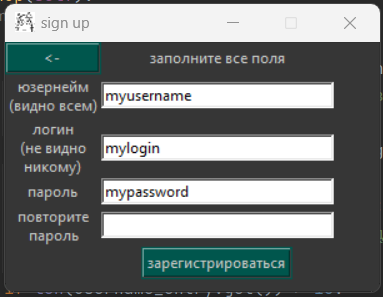


Рис. 2.2, 2.3 «Сообщения ‘заполните все поля’, ‘недопустимые символы (:)’»

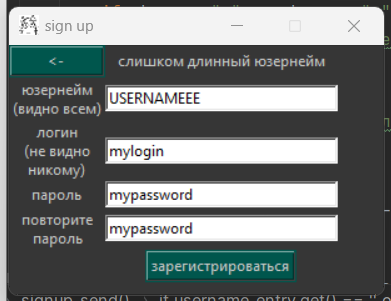
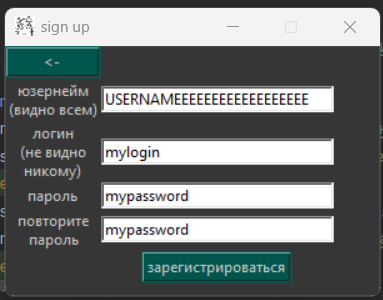


Рис 2.4 «слишком длинный юзернейм»

Если данные, введенные в поле «пароль» не совпадают с данными, введенными в поле «повторите пароль», после нажатия на кнопку «зарегистрироваться» на экран выведется сообщение «пароли не совпадают!» (Рис 2.5)

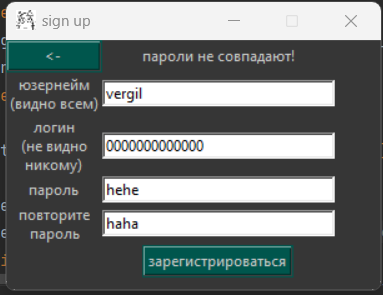


Рис 2.5 «пароли не совпадают!»

Если пользователь ввел в поле логина «login» или в поле пароля «password», после нажатия на кнопку «зарегистрироваться» на экран выведется сообщение «не-а» (Рис 2.6.). Это сделано затем, чтобы предотвратить ошибку при чтении базы данных, поскольку первый ряд таблицы содержит колонки именно с такими названиями.

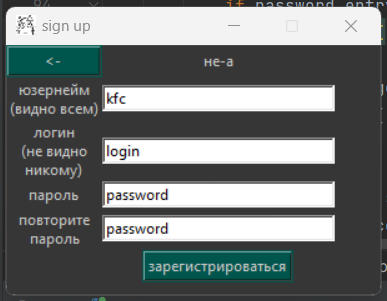


Рис 2.6 «не-а»

При успешной регистрации после нажатия на кнопку «зарегистрироваться» выводится сообщение с информацией, введенной пользователем, и открывается главное меню мессенджера (Рис. 2.7, Рис 4)

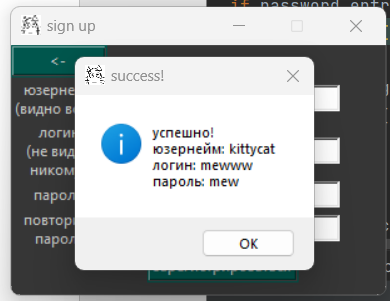


Рис 2.7 «успешная регистрация»  
  
 C:\Users\ajngc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\profile0.pngC:\Users\ajngc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\profile1.pngC:\Users\ajngc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\profile2.pngC:\Users\ajngc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\profile3.pngC:\Users\ajngc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\profile4.png

Рис 3.2 «Фотографии профилей»



Рис 3.1 «Иконка приложения»

Главное меню содержит в себе кнопки «настройки», «друзья», «сообщения», «помощь», «закрыть» (Рис. 4)

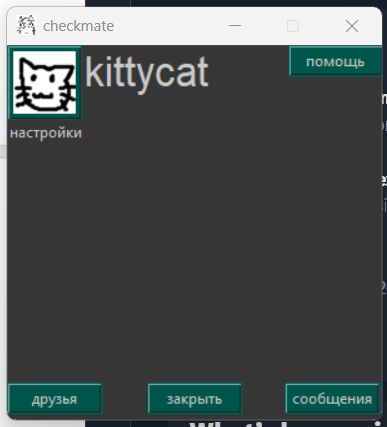


Рис. 4 «Главное меню»

При нажатии на кнопку «настройки» открывается окно настроек, содержащее кнопки «изменить юзернейм», «изменить фото», «изменить пароль», «выйти из аккаунта», «друзья», «сообщения», «главное меню» (Рис. 5.1). При повторном нажатии на кнопку настроек также открывается главное меню. При нажатии на кнопку «выйти из аккаунта» производится выход в окно входа в аккаунт.

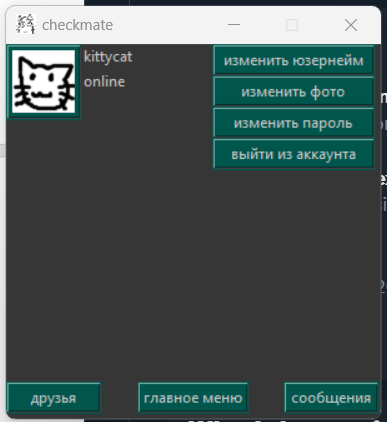


Рис. 5.1 «Окно настроек»

При нажатии на кнопку «изменить юзернейм» на месте юзернейма появляется поле ввода, в которое предлагается ввести новый (Рис. 5.2.1). По нажатию на кнопку «сохранить изменения» юзернейм обновляется.

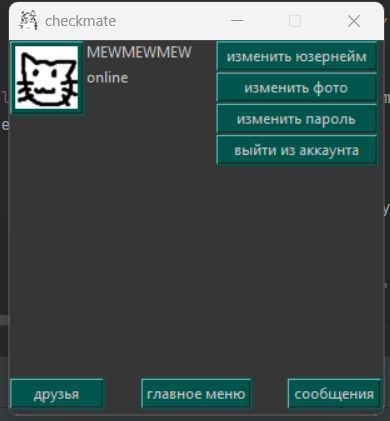
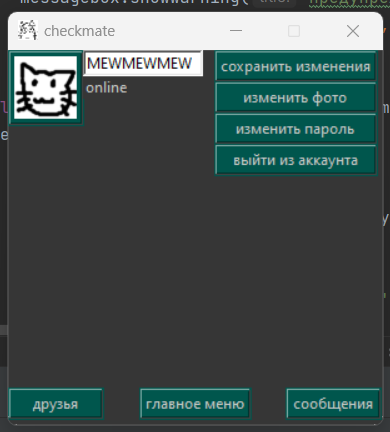
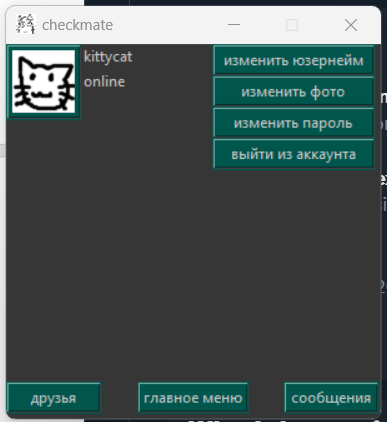
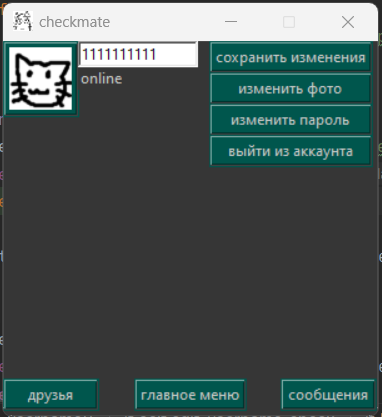
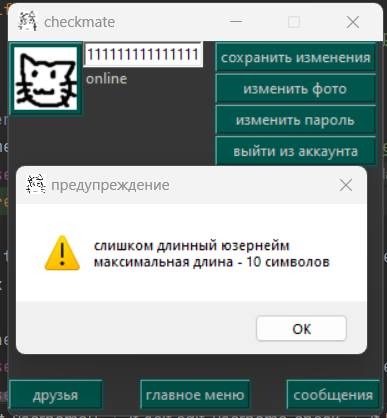


Рис. 5.2.1 «Изменение юзернейма»

Если длина введенного юзернейма превышает 10 символов, после нажатия на кнопку «сохранить изменения» появляется окно с сообщением «слишком длинный юзернейм! максимальная длина – 10 символов». Содержимое поля ввода обрезается до 10 символов (Рис. 5.2.2) Это нужно для того, чтобы юзернейм правильно отображался

Рис. 5.2.2 «слишком длинный юзернейм»

Если поле ввода пустое, после нажатия на кнопку «сохранить изменения» появляется окно с сообщением «юзернейм не может быть пустым» (Рис. 5.2.3)

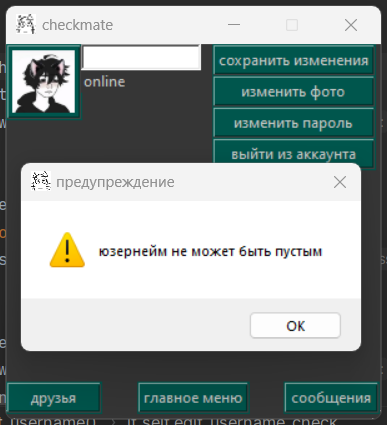


Рис. 5.2.3 «юзернейм не может быть пустым»

Если пользователь ввел в поле текст, содержащий символ «:», появляется окно с сообщением «недопустимые символы (:)» (Рис. 5.2.4)

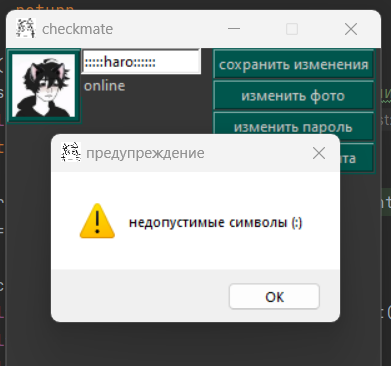


Рис. 5.2.4 «недопустимые символы (:)»

Если введенный пользователем юзернейм уже занят (или является словом «username»), появляется окно с сообщением «вы не можете использовать этот юзернейм:(» (Рис. 5.2.5)

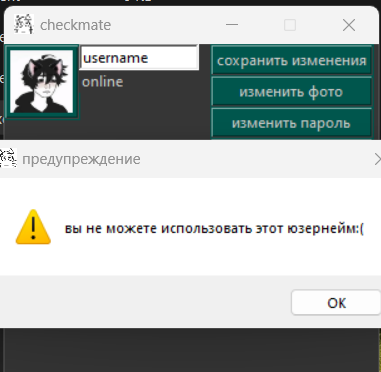
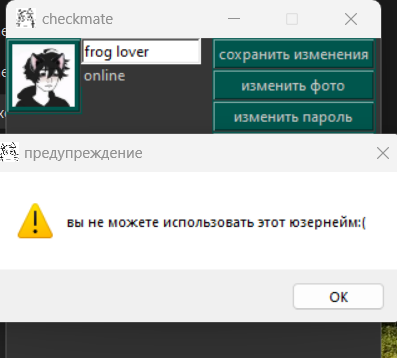


Рис. 5.2.5 «вы не можете использовать этот юзернейм:(»

При нажатии на кнопку «изменить фото» открывается окно с пятью кнопками, которые представляют собой разные картинки. По нажатию на кнопку фотография профиля будет изменена на соответствующую (Рис 5.3). Подробнее см. Рис. 3.2

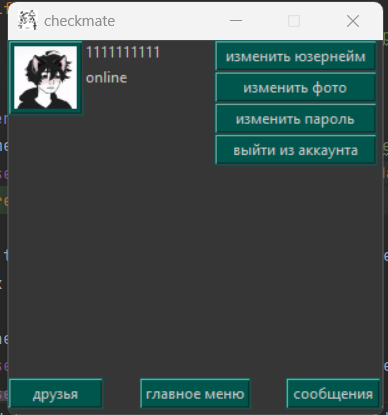
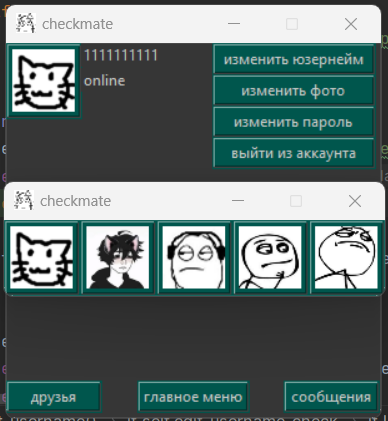


Рис. 5.3 «Изменение фотографии профиля»

При нажатии на кнопку «изменить пароль» открывается окно изменения пароля с полем ввода и кнопкой «сохранить изменения» (Рис. 5.4.1)

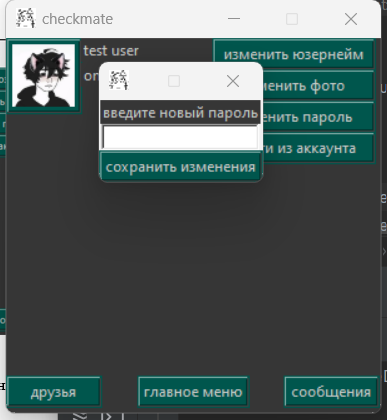


Рис. 5.4.1 «окно изменения пароля»

Если пользователь оставил поле пустым, появляется сообщение «пароль не может быть пустым». Если пользователь ввел «password» - «вы не можете использовать этот пароль». Если поле содержит символ «:» - «недопустимые символы (:)» (Рис. 5.4.2)

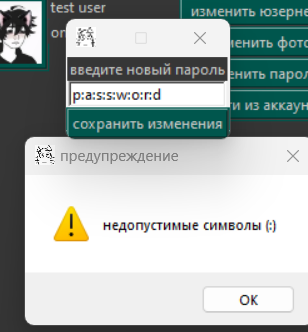
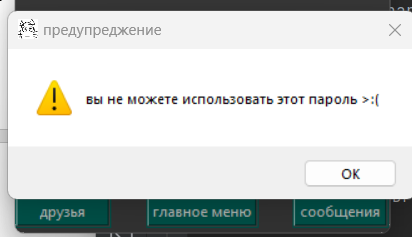
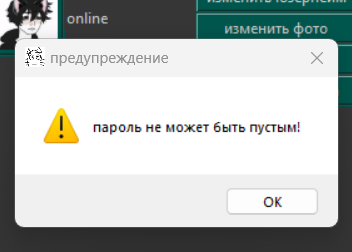


Рис. 5.4.2 «Сообщения при некорректном вводе нового пароля»

При нажатии на кнопку «друзья» открывается страница контактов (Рис. 6.1)

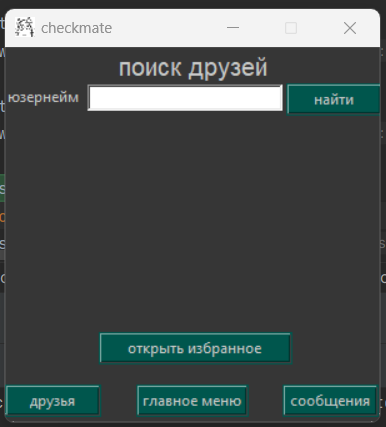


Рис. 6.1. «Страница контактов»

После нажатия на кнопку «найти» производится поиск пользователей в соответствии с данными, введенными в поле. Если пользователь не был найден, выведется «такой пользователь не существует». Если введенные данные совпадают с юзернеймом владельца аккаунта, выведется «вы нашли себя!». Если же пользователь найден был, выведется «пользователь найден», его имя и кнопка «отправить сообщение», при нажатии на которую открывается переписка с этим пользователем (Рис. 6.2)

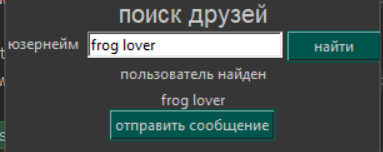
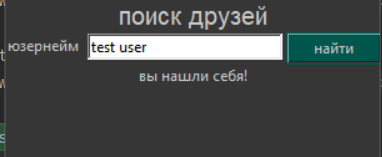
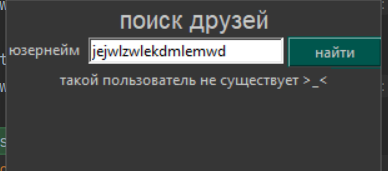
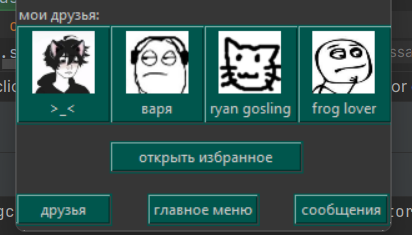
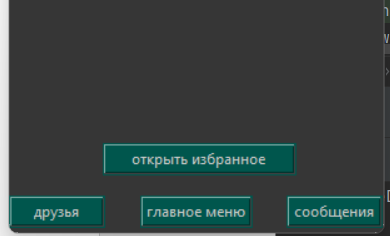
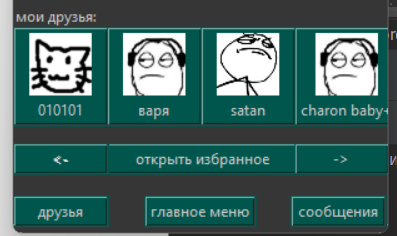


Рис. 6.2. «Поиск друзей»

Если владелец аккаунта еще ни разу не открывал переписку с другим пользователем, список контактов будет пуст. Если друзей меньше 5, будут показываться только они. Если друзей больше 5, появятся кнопки вправо и влево, по нажатию на которые можно листать контакты (Рис. 6.3)



  
Рис 6.3 «мои друзья»

По нажатию на кнопку «открыть избранное» открывается переписка с самим собой.

По нажатию на кнопку «сообщения» открывается последняя открытая переписка с момента открытия приложения. По умолчанию открывается переписка с самим собой. После выхода из приложения последняя переписка сбрасывается до значения по умолчанию. Отправление сообщений производится по нажатию клавиши enter.

По нажатию на кнопку «очистить» история сообщений очищается, причем очищается у обоих пользователей (Рис 7.1).

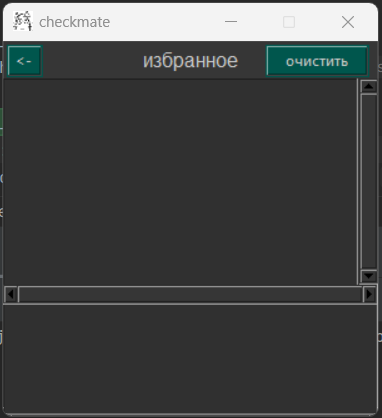
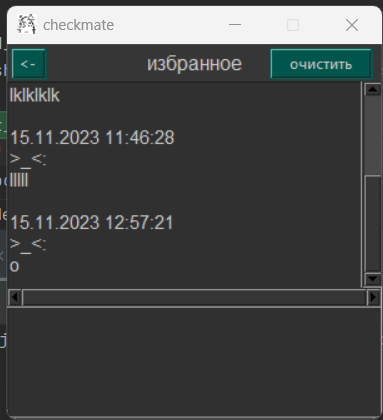


Рис. 7.1 «переписка; очистка истории сообщений»

При нажатии на кнопку «🡨» открывается главное меню.

При нажатии на кнопку «помощь» открывается окно, содержащее адрес электронной почты разработчика и кнопку «скопировать». (Рис. 8) При нажатии на нее адрес копируется в буфер обмена.

При нажатии на кнопку «закрыть» приложение закрывается.   
Если выход из аккаунта произведен не был, при повторном открытии приложения произведется автоматический вход в аккаунт и откроется главное меню.

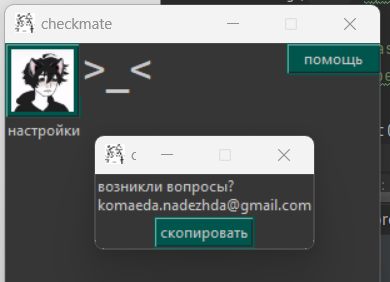


Рис. 8 «помощь»

# 4.2. Сервер

Сервер представляет собой консольное приложение, куда выводятся отчеты о полученных от клиента запросах и их обработке. (Рис. 9)

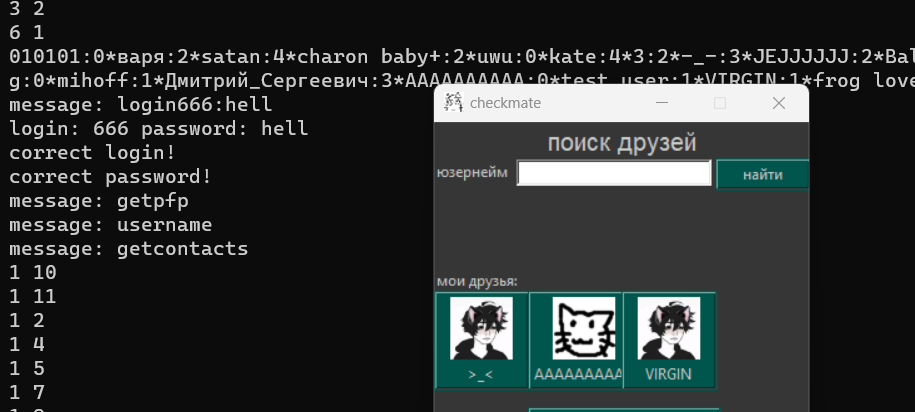


Рис. 9 «Интерфейс сервера»

5. Кодирование

# Создание сервера и подключение к серверу

Для обеспечения сетевого соединения я буду использовать модуль socket.

AF\_INET указывает, что сокет будет использовать семейство адресов IPv4. TCP (Transmission Control Protocol) и UDP (User Datagram Protocol) – два основных протокола передачи данных.

Протокол TCP обеспечивает надежную передачу данных: он гарантирует, что данные будут доставлены в том порядке, в котором были отправлены, и что ничего по пути потеряно не будет.

UPD обеспечивает быструю передачу данных, но не гарантирует, что они будут доставлены в том же порядке, в каком были отправлены, да и не гарантирует доставку данных вообще.

Поскольку мне нужно обеспечить надежное взаимодействие пользователей друг с другом и с базой данных, я буду использовать протокол TCP.

1. В функции main на сервере я получаю локальный IP-адрес компьютера, на котором он расположен, а затем указываю порт, на котором сервер будет прослушивать входящие соединения. Затем я создаю сокет и с помощью bind() привязываю его к этим IP-адресу и порту.
2. Метод listen() запускает сервер и начинает прослушивать входящие соединения. Затем с помощью цикла while True сервер ожидает подключения клиентов. Когда клиент подключается, вызывается метод accept(), который возвращает сокет клиента и адрес клиента.
3. Для обработки каждого подключения клиента с помощью модуля threading я создаю новый поток, запускающий функцию handle(), предназначенную для обработки получаемых запросов.

Листинг 1.1. Создание сервера

def main():  
 try:  
 host = socket.gethostbyname(socket.gethostname())  
 port = 8080  
 server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server.bind((host, port))  
 server.listen()  
 print('the server is online')  
  
 while True:  
 client, address = server.accept()  
  
 thread = threading.Thread(target=handle, args=(client, address))  
 thread.start()  
  
 except Exception as e:  
 print(e)

При запуске клиента сразу же производится попытка подключения к серверу: в классе messenger в конструкторе класса вызывается метод enterip(), где нужно ввести IP-адрес сервера. (Листинг 1.2)

Чтобы подключиться через глобальную сеть, нужно указать глобальный IP-адрес компьютера, на котором расположен сервер, а чтобы через локальную – соответственно, локальный.

Я создала возможность сохранения IP-адреса в случае успешного подключения для удобства: вводить его каждый раз при запуске приложения не придется.

Для начала вызывается вложенная функция instantip(), отвечающая за мгновенное подключение. Она проверяет, существует ли файл под названием ‘ip.txt’. Если существует, его содержимое записывается в переменную host. Если файл пуст, производится выход из функции. В противном случае вызывается метод establish\_connection(). Если файл не существует, то он создается, и также производится выход из функции.

Если мгновенное подключение не удалось, то создается окно с полем ввода, куда нужно ввести IP-адрес, и кнопкой, по нажатию на которую данные из поля ввода записываются в переменную host и в файл ‘ip.txt’, закрывается окно ввода IP-адреса и вызывается метод establishs\_connection(). Если поле пустое, то выводится предупреждение.

Листинг 1.2. Ввод IP-адреса

def enterip(self):  
 def instantip():  
 if os.path.exists('ip.txt'):  
 with open('ip.txt', 'r') as file:  
 self.host = file.read()  
 if self.host == '':  
 return  
 with open('ip.txt', 'w') as file:  
 file.write(self.host)  
 self.establish\_connection()  
 else:  
 with open('ip.txt', 'w'):  
 return  
  
 def savehostip():  
 self.host = ip\_entry.get()  
 if self.host == '':  
 messagebox.showwarning('предупреждение', 'заполните поле')  
 return  
 with open('ip.txt', 'w') as file:  
 file.write(self.host)  
 ip\_window.destroy()  
 self.establish\_connection()  
  
 instantip()  
  
 self.host = None  
 ip\_window = Tk()  
 screen\_width = ip\_window.winfo\_screenwidth()  
 screen\_height = ip\_window.winfo\_screenheight()  
 x = int((screen\_width / 2) - (150 / 2))  
 y = int((screen\_height / 2) - (80 / 2))  
 ip\_window.geometry("{}x{}+{}+{}".format(150, 80, x, y))  
 ip\_window.resizable(False, False)  
 ip\_window.title('checkmate')  
  
 ip\_label = ttk.Label(ip\_window, text='введите ip адрес сервера:')  
 ip\_entry = ttk.Entry(ip\_window)  
 ip\_button = ttk.Button(ip\_window, text='подключиться к серверу', command=savehostip)  
  
 ip\_label.pack()  
 ip\_entry.pack()  
 ip\_button.pack()  
  
 ip\_window.protocol("WM\_DELETE\_WINDOW", exit)  
 ip\_window.mainloop()

Метод establish\_connection() – подключение к серверу. (Листинг 1.3)

1. Задается значение порта, создается сокет, в котором указывается, что я буду использовать пространство адресов IPv4 и протокол TCP.
2. С помощью connect() производится подключение к серверу по заданному порту и полученному значению IP-адреса.
3. Если подключение прошло успешно, запускается метод входа в аккаунт. Если подключение не удалось, то содержимое файла ‘ip.txt’ очищается, и заново запускается функция self.enterip().

Листинг 1.3. Подключение клиента к серверу

def establish\_connection(self):  
 port = 8080  
 try:  
 self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.client.connect((self.host, port))  
 self.login()  
 except Exception:  
 with open('ip.txt', 'w'):  
 pass  
 self.enterip()

# Обмен данными

**Со стороны сервера:**

Для получения запросов сервером от клиента я использую функцию receive(). В нее передается аргумент client, который уточняет, с каким клиентом нужно взаимодействовать. (Листинг 2.1.). С ее помощью сервер принимает по 1024 байта, пока не появится символ “|”, означающий конец сообщения. После данные декодируются в UTF-8. Если в процессе клиент отключился, в консоль выводится сообщение об ошибке.

Полученные сообщения представляют собой текст, начинающийся с определенного кода запроса, за которым следуют данные, которые нужно обработать.

Листинг 2.1. Получение запросов от клиента

def receive(client):  
 try:  
 data = b""  
 while b"|" not in data:  
 msg = client.recv(1024)  
 if not msg:  
 break  
 data += msg  
 return data.decode('utf-8').strip("|")  
 except Exception as e:  
 print('receiving', e)

Для отправления данных клиенту я использую функцию transmit(), принимающую аргументы client и message. Она отправляет нужное сообщение, добавляя в конце символ “|”, означающий его окончание, и кодирует его в UTF-8. Если в процессе клиент отключился, в консоль выводится сообщение об ошибке. (Листинг 2.2)

Листинг 2.2. Отправление данных клиенту

def transmit(client, message):  
 try:  
 client.send((str(message)+'|').encode('utf-8'))  
 except Exception as e:  
 print('transmitting', e)

Работа с клиентом происходит в функции handle(), которая принимает аргументы client (сокет клиента) и address (IP-адрес клиента). (Листинг 2.3)

1. В консоль выводится IP-адрес подключившегося клиента.
2. Переменной connection\_lost присваивается значение False, а переменной ID – число 0. Первая переменная содержит в себе информацию о состоянии соединения, а вторая содержит в себе ID пользователя. Она будет использоваться в дальнейшем при работе с базой данных.
3. Пока переменная connection\_lost не примет значение True, будет вызываться функция communication(), которая обрабатывает запросы, отправляемые клиентом. В ней через функцию receive() сервер получает сообщение от клиента и обрабатывает его. Если сообщение не получено, переменная connection\_lost принимает значение False.

Листинг 2.3. Функция для работы с подключенным клиентом

def handle(client, address):  
 print(f'connected to {address}')  
 connection\_lost = False  
 ID = 0  
 def communication():

nonlocal connection\_lost  
 nonlocal ID  
 message = receive(client)  
 if not message:  
 connection\_lost = True  
 return  
 print('message:', message)

#далее производится обработка запросов  
#...   
 while not connection\_lost:  
 communication()

В клиенте используются аналогичные функции. (Листинг 2.4, 2.5) Если в процессе передачи или получения данных сервер отключился, появляется оповещающее об этом пользователя окно (см. Листинг 2.6.) и приложение закрывается.

Листинг 2.4. Получение запросов с сервера

def receive(self):  
 try:  
 data = b""  
 while b"|" not in data:  
 message = self.client.recv(1024)  
 if not message:  
 break  
 data += message  
 return data.decode('utf-8').strip("|")  
 except Exception as e:  
 self.offline\_popup()  
 exit()

Листинг 2.5. Отправление данных на сервер

def transmit(self, message):  
 try:  
 self.client.send((str(message) + '|').encode('utf-8'))  
 except Exception as e:  
 self.offline\_popup()  
 exit()

Чтобы оповестить пользователя о том, что сервер в данный момент оффлайн, появляется окно с соответствующим текстом и кнопкой “закрыть”.

Листинг 2.6. Метод, запускающий окно, оповещающее о том, что сервер не запущен

def offline\_popup(self):  
 offline = Tk()  
 offline.geometry('150x100')  
 offline.resizable(False, False)  
 offline.title('checkmate')  
 offline\_label = ttk.Label(offline, text='сервер оффлайн >\_<')  
 exit\_button = ttk.Button(offline, text='закрыть', command=exit)  
 offline\_label.place(relx=0.5, rely=0.3, anchor='center')  
 exit\_button.place(relx=0.5, rely=0.8, anchor='center'  
 offline.mainloop()

# Вход в аккаунт

**Со стороны клиента:**

Для входа в аккаунт я использую метод login(). (см. Листинг 3.1.)

Сначала производится проверка возможности мгновенного входа с помощью вложенной в этот метод функции instant\_login\_check(). (см. Листинг 3.2.)

1. Проверяется существование файла под названием «details.txt», где содержатся логин и пароль пользователя. Если такой файл не существует, он создается и производится выход из функции.
2. Если такой файл существует, его содержимое записывается в переменную contents. Если же его содержимое пусто, производится выход из функции и запускается процесс ручного входа в аккаунт.
3. Если же в файле есть какие-то данные, то производится попытка разбить содержимое по символу “:”. В случае генерации ошибки файл очищается и производится выход из функции. Это нужно для того, чтобы ничего не ломалось, если пользователь изменил файл “details.txt”.
4. Если же разбить содержимое получилось, вызывается вложенная в метод login() функция мгновенного входа в аккаунт instant\_login(), в которую передаются полученные логин и пароль.

Листинг 3.1. Метод входа в аккаунт

def login(self):

#login\_password\_check()  
#redirect()  
#instant\_login()  
#instant\_login\_check()  
  
 instant\_login\_check()  
 login\_window = Tk()  
 login\_window.title('login')  
 try:  
 login\_icon = PhotoImage(master=login\_window, file='icon.png')  
 login\_window.iconphoto(True, login\_icon)  
 except TclError:  
 login\_window.withdraw()  
 messagebox.showwarning('ошибка', 'мессенджер не может быть запущен, потому что файлы приложения были изменены. переустановите приложение или обратитесь за помощью к разработчику\nkomaeda.nadezhda@gmail.com')  
 exit()  
  
 screen\_width = login\_window.winfo\_screenwidth()  
 screen\_height = login\_window.winfo\_screenheight()  
 x = int((screen\_width / 2) - (200 / 2))  
 y = int((screen\_height / 2) - (100 / 2))  
 login\_window.geometry("{}x{}+{}+{}".format(200, 100, x, y))  
 login\_window.resizable(False, False)  
  
 style1 = ttk.Style()  
 style1.theme\_use('alt')  
 style1.configure('TLabel', background="#363636", foreground='#cacaca')  
 style1.configure('TButton', background='#00564d', foreground="#cacaca")  
 style1.map('TButton', background=[('active', '#00897b')], foreground=[('active', '#cacaca')])  
 login\_window.configure(bg='#363636')  
  
 wrong\_label = ttk.Label(login\_window, text='', anchor="center", justify="center")  
 login\_label = ttk.Label(login\_window, text='логин')  
 password\_label = ttk.Label(login\_window, text='пароль')  
 login\_entry = ttk.Entry(login\_window, width=23)  
 password\_entry = ttk.Entry(login\_window, show='\*', width=23)  
 login\_button = ttk.Button(login\_window, text='войти', command=lambda: login\_password\_check(None))  
 signup\_button = ttk.Button(login\_window, text='зарегистрироваться', command=redirect)  
  
 wrong\_label.place(relx=0.5, rely=0.1, anchor='center')  
 login\_label.place(relx=0, rely=0.2)  
 login\_entry.place(relx=0.25, rely=0.2)  
 password\_label.place(relx=0, rely=0.42)  
 password\_entry.place(relx=0.25, rely=0.42)  
 (login\_button.place(relx=0.6, rely=0.7))  
 signup\_button.place(relx=0, rely=0.7)  
  
 login\_window.bind("<Return>", login\_password\_check)  
 login\_window.protocol("WM\_DELETE\_WINDOW", self.close\_window)  
 login\_window.mainloop()

Листинг 3.2. Функция проверки возможности мгновенного входа в аккаунт

def instant\_login\_check():  
 if os.path.exists('details.txt'):  
 with open('details.txt', 'r') as file:  
 contents = file.read()  
 if contents == '':  
 return  
 else:  
 try:  
 login, password = contents.split(':')  
 instant\_login(login, password)  
 except Exception:  
 with open('details.txt', 'w'):  
 pass  
 return  
 else:  
 with open('details.txt', 'w'):  
 return

Вложенная функция instant\_login производит мгновенный вход в аккаунт без необходимости ввода данных вручную. (см. Листинг 3.3)

1. Производится проверка корректности полученных данных. Если в логине или пароле содержатся символы “:” или “\*”, или же их значения совпадают с “login” и “password”, производится выход из функции. Первая проверка нужна потому, что сервер использует символы “:” и “\*” для разграничения полученных данных. Вторая нужна для исключения ошибки чтения сервером базы данных, поскольку первый ее ряд содержит колонки именно с такими названиями.
2. Далее, в случае успешного прохождения проверок, на сервер отправляется одно сообщение, начинающееся с кода запроса - ‘login’, за которым идут логин и пароль, разделенные знаком “:”.
3. Далее ожидается ответ от сервера. Ответ «1», значит то, что совпадение в базе данных найдено и вход в аккаунт произведен успешно. Открывается главное окно приложения.
4. Ответ «2» означает, что совпадение в базе данных не найдено. Содержимое файла “details.txt” очищается, и производится выход из функции. Если ответ не получен, открывается окно, оповещающее пользователя о том, что сервер сейчас оффлайн (См. Листинг 2.5.)

Листинг 3.3. Функция мгновенного входа в аккаунт

def instant\_login(login, password):  
 for data in [login, password]:  
 for char in data:  
 if char == ":" or char == "\*":  
 return  
 if login == 'login' or password == 'password':  
 return  
 self.transmit(f'login{login}:{password}')  
 response = self.receive()  
 if response == "1":  
 self.main\_info()  
 self.main\_menu\_place()  
 self.root.mainloop()  
 return  
 elif response == '2':  
 with open('details.txt', 'w'):  
 pass  
 self.login()  
 else:  
 self.offline\_popup()  
 exit()

Если мгновенный вход в аккаунт не удался, открывается окно, в котором предлагается произвести вход в аккаунт, введя данные вручную. (См. Листинг 3.1.)

1. Создается окно tkinter и производится проверка целостности файлов программы (фотографий). Это необходимо, потому что они могут быть изменены или удалены пользователем.
2. Задается размер окна, его расположение.
3. Создается стиль, задается тема и конфигурируется вид виджетов.
4. Создаются и располагаются виджеты.
5. К клавише enter и кнопке «войти» привязывается вызов вложенной функции login\_password\_check() (см. Листинг 3.4).
6. К кнопке «зарегистрироваться» привязывается вызов вложенной функции redirect() (см. Листинг 3.5)
7. Когда пользователь нажимает на кнопку закрытия окна на верхней панели, вызывается метод close\_window.

Для входа в аккаунт используется вложенная функция login\_password\_check() (см. Листинг 3.4.)

1. Производится проверка корректности полученных данных. Если в логине или пароле содержатся символы “:” или “\*”, или же их значения совпадают с “login” и “password”, производится выход из функции. Первая проверка нужна потому, что сервер использует символы “:” и “\*” для разграничения полученных данных. Вторая нужна для исключения ошибки чтения сервером базы данных, поскольку первый ее ряд содержит колонки именно с такими названиями.
2. Далее, в случае успешного прохождения проверок, на сервер отправляется одно сообщение, начинающееся с кода запроса - ‘login’, за которым идут логин и пароль, разделенные знаком “:”.
3. Далее ожидается ответ от сервера. Ответ «1», значит то, что совпадение в базе данных найдено и вход в аккаунт произведен успешно. Логин и пароль записываются в файл “details.txt”. Окно входа в аккаунт закрывается, и открывается главное окно приложения.
4. Ответ «2» означает, что совпадение в базе данных не найдено. Выводится соответствующее сообщение. Если ответ не получен, открывается окно, оповещающее пользователя о том, что сервер сейчас оффлайн (См. Листинг 2.6.)

Листинг 3.4. Функция входа в аккаунт

def login\_password\_check(event):  
 for data in [login\_entry.get(), password\_entry.get()]:  
 for char in data:  
 if char == ":" or char == "\*":  
 wrong\_label['text'] = 'недопустимые символы (:)'  
 return  
  
 if login\_entry.get() == 'login' or password\_entry.get() == 'password':  
 wrong\_label['text'] = 'не-а'  
 return  
  
 self.transmit('login'+str(login\_entry.get())+':'+str(password\_entry.get()))  
 response = self.receive()  
 if response.startswith('1'):  
 with open('details.txt', 'w') as file:  
 file.write(login\_entry.get()+':'+password\_entry.get())  
  
 login\_window.destroy()  
 self.main\_info()  
 self.main\_menu\_place()  
 self.root.mainloop()  
 elif response == '2':  
 wrong\_label['text'] = '>\_< некорректные данные'  
 else:  
 wrong\_label['text'] = '>\_< что-то не так'  
 self.offline\_popup()  
 exit()

При нажатии на кнопку «зарегистрироваться», вызывается вложенная функция redirect() (см. Листинг 3.5), которая закрывает окно входа в аккаунт и осуществляет переход в окно регистрации.

Листинг 3.5. Функция перехода в окно регистрации

def redirect():  
 login\_window.destroy()  
 self.signup()

**Со стороны сервера:**

При попытке входа в аккаунт на сервер отправляется сообщение, начинающееся со слова ‘login’. Функция communication() ищет совпадение: если полученное сообщение начинается со слова ‘login’, выполняется следующий код (см. Листинг 3.6.)

1. Сам код запроса обрезается (в данном случае это ‘login’), и остаются только логин и пароль, которые разбиваются по символу «:».
2. Далее производится чтение базы данных. Если найден полученный логин, проверяется, соответствует ли ему полученный пароль. Если да, то переменной ID присваивается значение поля ID пользователя в базе данных, и клиенту отправляется подтверждение успешной проверки – сообщение «1».
3. В противном случае клиенту отправляется сообщение «2», означающее то, что совпадение не найдено и вход в аккаунт не произведен.
4. В консоль выводятся отчеты.

Листинг 3.6. Обработка кода запроса «login»  
if message.startswith('login'):  
 message = message[len('login'):]  
 login, password = message.split(':')  
 print('login:', login, 'password:', password)  
 for cell in sheet['B']:  
 if login == str(cell.value):  
 print('correct login!')  
 if password == cell.offset(column=1).value:  
 print('correct password!')  
 ID = int(cell.offset(column=-1).value)  
 transmit(client, '1')  
 print('successful login')  
 return  
 else:  
 print('wrong login or password')  
 transmit(client, '2')

# Регистрация пользователей

**Со стороны клиента:**

Для регистрации пользователей я использую метод signup() (Листинг 4.1), в котором открывается окно с полями ввода «юзернейм», «логин», «пароль» и «повторите пароль». По нажатию на клавишу enter или кнопку «send» вызывается вложенная функция signup\_send() (Листинг 4.2.) По нажатию на клавишу «🡨» производится переход обратно в окно входа в аккаунт (Листинг 4.3)

Листинг 4.1. Метод регистрации пользователей.

def signup(self):  
#signup\_send()  
#go\_back()  
 signup\_window = Tk()  
 signup\_window.title('sign up')  
 login\_icon = PhotoImage(master=signup\_window, file='icon.png')  
 signup\_window.iconphoto(True, login\_icon)  
  
 screen\_width = signup\_window.winfo\_screenwidth()  
 screen\_height = signup\_window.winfo\_screenheight()  
 x = int((screen\_width / 2) - (280 / 2))  
 y = int((screen\_height / 2) - (150 / 2))  
 signup\_window.geometry("{}x{}+{}+{}".format(300, 200, x, y))  
  
 signup\_window.resizable(False, False)  
  
 style1 = ttk.Style()  
  
 style1.theme\_use('alt')  
 style1.configure('TLabel', background="#363636", foreground='#cacaca')  
 style1.configure('TButton', background='#00564d', foreground="#cacaca")  
 style1.map('TButton', background=[('active', '#00897b')], foreground=[('active', '#cacaca')])  
 signup\_window.configure(bg='#363636')  
  
 wrong\_label = ttk.Label(signup\_window, anchor="center", justify="center")  
 back\_button = ttk.Button(signup\_window, text='<-', command=go\_back)  
 username\_label = ttk.Label(signup\_window, text='юзернейм\n(видно всем)', anchor="center", justify="center")  
 username\_entry = ttk.Entry(signup\_window, width=30)  
 login\_label = ttk.Label(signup\_window, text='логин\n(не видно\nникому)', anchor="center", justify="center")  
 login\_entry = ttk.Entry(signup\_window, width=30)  
 password\_label1 = ttk.Label(signup\_window, text='пароль')  
 password\_entry1 = ttk.Entry(signup\_window, width=30)  
 password\_label2 = ttk.Label(signup\_window, text='повторите\n пароль')  
 password\_entry2 = ttk.Entry(signup\_window, width=30)  
 signup\_button = ttk.Button(signup\_window, text='зарегистрироваться', command=lambda: signup\_send(None))  
  
 wrong\_label.grid(row=0, column=1)  
 back\_button.grid(row=0, column=0)  
 username\_label.grid(row=1, column=0)  
 username\_entry.grid(row=1, column=1)  
 login\_label.grid(row=2, column=0)  
 login\_entry.grid(row=2, column=1)  
 password\_label1.grid(row=3, column=0)  
 password\_entry1.grid(row=3, column=1)  
 password\_label2.grid(row=4, column=0)  
 password\_entry2.grid(row=4, column=1)  
 signup\_button.grid(row=5, column=1)  
  
 signup\_window.bind("<Return>", signup\_send)  
  
 signup\_window.mainloop()

В функции signup\_send() (Листинг 4.2) производятся разнообразные проверки: для начала проверяется, заполнены ли все поля, содержатся ли в них недопустимые символы, превышает ли поле «юзернейм» допустимую длину», совпадают ли логин и пароль со значениями «login» и «password».

В случае успешного прохождения проверок полученные из полей ввода данные отправляются на сервер одним сообщением, которое начинается с кода запроса «signup», за которым через символ “:” следуют юзернейм, логин и пароль.

В зависимости от ответа сервера на экран выводятся сообщения о том, что введенные логин или юзернейм заняты, или, в случае успешной регистрации, выводится окно с введенными пользователями, чтобы он мог их себе куда-нибудь сохранить, и открывается главное окно мессенджера.

Листинг 4.2. Отправление данных о регистрации на сервер

def signup\_send(event):  
 if username\_entry.get() == '' or login\_entry.get() == '' or password\_entry1.get() == '' or password\_entry2.get() == '':  
 wrong\_label['text'] = 'заполните все поля'  
 return  
 for data in [username\_entry.get(), login\_entry.get(), password\_entry1.get(), password\_entry2.get()]:  
 for char in data:  
 if char == ":" or char == "\*":  
 wrong\_label['text'] = 'недопустимые символы (:)'  
 return  
 if len(username\_entry.get()) > 10:  
 wrong\_label['text'] = 'слишком длинный юзернейм'  
 username\_entry.delete('10', 'end')  
 return  
 if password\_entry1.get() != password\_entry2.get():  
 wrong\_label['text'] = 'пароли не совпадают!'  
 return  
 if login\_entry.get() == 'login' or password\_entry1.get() == 'password':  
 wrong\_label['text'] = 'не-а'  
 return  
  
self.transmit(f'signup{username\_entry.get()}:{login\_entry.get()}:{password\_entry1.get()}')  
 check = self.receive()  
 if check == '1':  
 messagebox.showinfo('success!', f'успешно!\nюзернейм: {username\_entry.get()}\nлогин: {login\_entry.get()}\nпароль: {password\_entry1.get()}')  
 with open('details.txt', 'w') as file:  
 file.write(f'{login\_entry.get()}:{password\_entry1.get()}')  
 go\_back()  
 elif check == '2':  
 wrong\_label['text'] = 'этот юзернейм уже занят'  
 else:  
 wrong\_label['text'] = 'этот логин уже занят'

Листинг 4.3. Переход в окно входа в аккаунт

def go\_back():  
 signup\_window.destroy()  
 self.login()

**Со стороны сервера:**

Если сообщение начинается со слова «signup» (Листинг 4.5), то кодовое слово обрезается, и оставшаяся часть сообщения разбивается по символу “:”. Так мы получаем юзернейм, логин и пароль, которые ввел пользователь.

Далее проверяется уникальность юзернейма и логина: D и B – это индексы колонок в таблице Excel, которые содержат соответствующие данные. Если проверки не пройдены, клиенту отправляется соответствующий код ошибки. Если проверки пройдены, клиенту отправляется код «1», который сигнализирует о том, что регистрация прошла успешно. В переменную ID сохраняется ID пользователя, полученный путем вычисления длины колонки под индексом «А» (длина = количество существующих пользователей). Юзернейм, логин и пароль, ID и индекс фотографии профиля (0 по умолчанию) добавляются в базу данных. Изменения сохраняются.

Листинг 4.5. Обработка кода запроса «signup»

elif message.startswith('signup'):  
 username, login, password = message[len('signup'):].split(':')  
 print(username, login, password)  
 for cell in sheet['D']:  
 if cell.coordinate != f'D{ID+1}' and username == str(cell.value):  
 transmit(client, 2)  
 print('username already exists')  
 return  
 for cell in sheet['B']:  
 if cell.coordinate != f'B{ID+1}' and login == str(cell.value):  
 transmit(client, 3)  
 print('login already exists')  
 return  
  
 transmit(client, 1)  
 print('successful signup')  
 ID = len(sheet['A'])  
 sheet[f'A{ID+1}'] = ID  
 sheet[f'B{ID+1}'] = login  
 sheet[f'C{ID+1}'] = password  
 sheet[f'D{ID+1}'] = username  
 sheet[f'E{ID+1}'] = 0  
 database.save("database.xlsx")

# Основные виджеты и переменные главного окна мессенджера

**Со стороны клиента:**

В методе main\_info() (Листинг 6.1) создаются основные виджеты и переменные.

1. С помощью ttk.Style() вместо дефолтных я устанавливаю собственные цвета для виджетов.
2. Я задаю переменным profile\_click\_check, contacts\_click\_check и edit\_username\_check значение False. Эти переменные нужны для правильного отображения виджетов соответствующих окон.
3. Создаются переменные profile\_logo, содержащие в себе фотографии профиля. Если эти файлы повреждены/изменены, выводится сообщение об ошибке, и приложение закрывается.
4. На сервер отправляется запрос «getpfp», мы получаем индекс установленной фотографии профиля для этого аккаунта, и присваиваем переменной profile\_var соответствующую картинку.
5. На сервер отправляется запрос «username», и мы присваиваем переменной username полученное значение.
6. Переменной friend\_username присваивается значение «избранное». Эта переменная используется для отображения имени собеседника, а по дефолту при нажатии на кнопку «сообщения» открывается переписка с самим собой. Для этой цели и присваиваем этой переменной такое значение, а также переменной friend\_ID значение None.
7. Создаются кнопки главного окна, им присваиваются методы перехода на соответствующие страницы.
8. По нажатии на кнопку «помощь» вызывается вложенная функция help(), которая открывает окошко, содержащее почту разработчика и кнопку «скопировать», которая ее копирует.

Листинг 6.1. Создание главного окна

def main\_info(self):  
 def help():  
 def copy():  
 help\_window.clipboard\_clear()  
 help\_window.clipboard\_append('komaeda.nadezhda@gmail.com')  
  
 help\_window = Toplevel(self.root)  
 width = help\_window.winfo\_reqwidth()  
 height = help\_window.winfo\_reqheight()  
 x = int((help\_window.winfo\_screenwidth() / 2 - width / 2))  
 y = int((help\_window.winfo\_screenheight() / 2 - height / 2))  
 help\_window.geometry("+{}+{}".format(x, y))  
 help\_window.grab\_set()  
 help\_window.resizable(False, False)  
 help\_window.configure(bg='#363636')  
  
 info\_label = ttk.Label(help\_window, text='возникли вопросы?\nkomaeda.nadezhda@gmail.com')  
 copy\_button = ttk.Button(help\_window, text='скопировать', command=copy)  
 info\_label.pack()  
 copy\_button.pack()

self.root = Tk()  
 self.root.geometry('300x300')  
 self.root.title('checkmate')  
 main\_icon = PhotoImage(master=self.root, file='icon.png')  
 self.root.iconphoto(True, main\_icon)  
 self.root.resizable(False, False)  
 screen\_width = self.root.winfo\_screenwidth()  
 screen\_height = self.root.winfo\_screenheight()  
 x = int((screen\_width / 2) - (300 / 2))  
 y = int((screen\_height / 2) - (300 / 2))  
 self.root.geometry("{}x{}+{}+{}".format(300, 300, x, y))  
  
 dark\_theme = ttk.Style()  
 dark\_theme.theme\_use('alt')  
 dark\_theme.configure('TLabel', background="#363636", foreground='#cacaca')  
 dark\_theme.configure('TButton', background='#00564d', foreground="#cacaca")  
 dark\_theme.map('TButton', background=[('active', '#00897b')], foreground=[('active', '#cacaca')])  
 dark\_theme.configure('TScrollbar', background='#363636', troughcolor='#4a4a4a')  
 dark\_theme.map('TScrollbar', background=[('active', '#363636')], troughcolor=[('active', '#4a4a4a')])  
 self.root.configure(bg='#363636')  
  
 self.profile\_click\_check = False  
 self.contacts\_click\_check = False  
 self.edit\_username\_check = False

try:  
 self.profile\_logo0 = PhotoImage(master=self.root, file='profile0.png')  
 self.profile\_logo1 = PhotoImage(master=self.root, file='profile1.png')  
 self.profile\_logo2 = PhotoImage(master=self.root, file='profile2.png')  
 self.profile\_logo3 = PhotoImage(master=self.root, file='profile3.png')  
 self.profile\_logo4 = PhotoImage(master=self.root, file='profile4.png')  
 except TclError:  
 self.root.withdraw()  
 messagebox.showwarning('ошибка', 'мессенджер не может быть запущен, потому что фотографии были изменены. переустановите приложение или обратитесь за помощью к разработчику\nkomaeda.nadezhda@gmail.com')  
 exit()  
  
 self.transmit('getpfp')  
 pfp\_index = self.receive()  
 self.profile\_var = self.profile\_logo0  
 if pfp\_index == '0':  
 self.profile\_var = self.profile\_logo0  
 elif pfp\_index == '1':  
 self.profile\_var = self.profile\_logo1  
 elif pfp\_index == '2':  
 self.profile\_var = self.profile\_logo2  
 elif pfp\_index == '3':  
 self.profile\_var = self.profile\_logo3  
 elif pfp\_index == '4':  
 self.profile\_var = self.profile\_logo4  
  
 self.transmit('username')  
 self.username = StringVar()  
 self.username.set(self.receive())  
 print(self.username.get())  
  
 self.friend\_username = StringVar(value='избранное')

self.profile\_button = ttk.Button(self.root, image=self.profile\_var, command=self.profile\_click)  
 self.profile\_label = ttk.Label(self.root, text='настройки')

self.contacts\_button = ttk.Button(self.root, text="друзья", command=self.contacts\_click)

self.friend\_ID = None  
 self.messages\_button = ttk.Button(self.root, text="сообщения", command=lambda: self.messages\_click(self.friend\_ID))  
  
 self.exit\_button = ttk.Button(self.root, text='закрыть', command=self.close\_window)  
 self.greeting = ttk.Label(self.root, textvariable=self.username, font=font.Font(size=25))  
 self.help\_button = ttk.Button(self.root, text='помощь', command=help)  
  
 self.root.protocol("WM\_DELETE\_WINDOW", self.close\_window)

**Со стороны сервера:**

Если сервер получил сообщение, начинающееся с кода запроса «username», клиенту передается юзернейм, найденный в базе данных по его ID. (Листинг 6.2)

Листинг 6.2. Обработка кода запроса «username»

elif message.startswith('username'):  
 transmit(client, sheet[f'D{ID+1}'].value)  
 print('username sent')

Если сервер получил сообщение, начинающееся с кода запроса «getpfp», клиенту отправляется индекс фотографии профиля, найденный в базе данных по его ID. (Листинг 6.3)

Листинг 6.3. Обработка кода запроса «getpfp»

elif message.startswith('getpfp'):  
 index = sheet[f'E{ID+1}'].value  
 transmit(client, index)  
 print('pfp sent')

# Настройки аккаунта

**Со стороны клиента:**

Метод profile\_click() создает страницу настроек. (Листинг 7.1)

1. Создаются нужные для этого окна виджеты:

* «изменить юзернейм»: по нажатию на эту кнопку вызывается вложенная функция edit\_username() (Листинг 7.2)
* «изменить пароль»: по нажатию на эту кнопку вызывается вложенная функция edit\_password() (Листинг 7.3)
* «изменить фото»: по нажатию на эту кнопку вызывается вложенная функция edit\_pfp() (Листинг 7.4)
* «выйти из аккаунта»: по нажатию на эту кнопку вызывается вложенная функция log\_out() (Листинг 7.5)
* «закрыть»: по нажатию на эту кнопку приложение закрывается

1. Переменная profile\_click\_check используется для отслеживания текущего состояния страницы настроек. Если она равна False, то это означает, что страница настроек не отображается. В этом случае виджеты, не принадлежащие к этой странице, скрываются, а принадлежащие, наоборот, размещаются. Если она равна True, размещаются виджеты главного меню. Это сделано для того, чтобы можно было открывать и закрывать эту страницу по нажатию на одну и ту же кнопку.

Листинг 7.1. Страница настроек

def profile\_click(self):

#edit\_username()  
#edit\_password()  
#edit\_pfp()  
#log\_out()  
  
 self.username\_label = ttk.Label(self.root, textvariable=self.username, anchor='center')  
 self.online\_label = ttk.Label(self.root, text='online')  
 self.edit\_username\_button = ttk.Button(self.root, text='изменить юзернейм', command=edit\_username, width=20)  
 self.username\_entry = ttk.Entry(self.root, textvariable=self.username, width=15)  
 self.edit\_password\_button = ttk.Button(self.root, text='изменить пароль', command=edit\_password, width=20)  
 self.edit\_pfp\_button = ttk.Button(self.root, text='изменить фото', command=edit\_pfp, width=20)  
 self.logout\_button = ttk.Button(self.root, text='выйти из аккаунта', command=log\_out, width=20)  
  
 if self.profile\_click\_check:  
 self.profile\_click\_check = False  
 self.main\_menu\_place()  
 else:  
 self.help\_button.place\_forget()  
 self.greeting.place\_forget()  
 self.profile\_label.place\_forget()  
 self.username\_label.place(x=60, y=0)  
 self.online\_label.place(x=60, y=20)  
 self.edit\_username\_button.place(relx=0.55, y=0)  
 self.edit\_pfp\_button.place(relx=0.55, y=25)  
 self.edit\_password\_button.place(relx=0.55, y=50)  
 self.logout\_button.place(relx=0.55, y=75)  
 self.exit\_button.configure(command=self.main\_menu\_place, text='главное меню')  
 self.profile\_click\_check = True

Вложенная функция edit\_username() (Листинг 7.2) позволяет изменить юзернейм.

Переменная edit\_username\_check нужна для корректного отображения виджетов, задействованных в этой функции.

1. Если edit\_username\_check равна False, вместо лейбла с юзернеймом появляется поле ввода, текст кнопки «изменить юзернейм» меняется на «сохранить изменения» и значение edit\_username\_check изменяется на True.
2. Если edit\_username\_check равна True, мы получаем данные из поля ввода и производим следующие проверки: на пустое поле ввода, на наличие недопустимых символов, на превышение допустимой длины юзернейма. Если проверки не пройдены, выводится предупреждение с соответствующим текстом.  
   Если проверки успешно пройдены, на сервер отправляется одно сообщение, начинающееся с кода запроса «editusername», за которым идет полученный из поля ввода текст.  
   Если сервер прислал ответ «1», это значит, что совпадений в базе данных не найдено: этот юзернейм уникален. Переменной username присваивается новый юзернейм, поле ввода пропадает, и на его месте появляется лейбл с новым юзернеймом. Текст кнопки также изменяется на свое предыдущее значение. Переменная self\_username\_check принимает значение False.  
   Если сервер прислал иной ответ, то выводится предупреждение, оповещающее о том, что такой юзернейм использовать нельзя.

Листинг 7.2. Изменение юзернейма

def edit\_username():  
 if self.edit\_username\_check:  
 if self.username\_entry.get() == '':  
 messagebox.showwarning("предупреждение", "юзернейм не может быть пустым")  
 return  
  
 for char in self.username\_entry.get():  
 if char == ":" or char == "\*":  
 messagebox.showwarning("предупреждение", "недопустимые символы (:)")  
 return  
  
 if len(self.username\_entry.get()) > 10:  
 messagebox.showwarning("предупреждение", "слишком длинный юзернейм\nмаксимальная длина - 10 символов")  
 self.username\_entry.delete('10', 'end')  
 return  
  
 self.transmit(f'editusername{self.username\_entry.get()}')  
 check = self.receive()  
  
 if check == '1':  
 self.username.set(self.username\_entry.get())  
 self.username\_entry.place\_forget()  
 self.username\_label.place(x=60, y=0)  
 self.edit\_username\_button['text'] = 'изменить юзернейм'  
 self.edit\_username\_check = False  
 else:  
 messagebox.showwarning("предупреждение", "вы не можете использовать этот юзернейм:(")  
  
 else:  
 self.username\_label.place\_forget()  
 self.username\_entry.place(x=60, y=0)  
 self.edit\_username\_button['text'] = 'сохранить изменения'  
 self.edit\_username\_check = True

Вложенная функция edit\_password() (Листинг 7.3) позволяет пользователю изменить пароль.

1. Поверх основного окна появляется новое, содержащее поле ввода и кнопку «сохранить изменения», по нажатию на которую вызывается вложенная функция save\_password()
2. Функция save\_password() проводит проверки на пустое поле, наличие недопустимых символов и совпадение введенных данных со словом “password”.   
   Если проверки не пройдены, появляется предупреждение с соответствующим сообщением.   
   Если проверки пройдены, на сервер передается одно сообщение, начинающееся с кода запроса «editpassword», за которым следует полученный из поля ввода текст. Окно изменения пароля закрывается.

Листинг 7.3. Изменение пароля

def edit\_password():  
 def save\_password():  
 if edit\_password\_entry.get() == '':  
 messagebox.showwarning("предупреждение", "пароль не может быть пустым!")  
 return  
 if edit\_password\_entry.get() == 'password':  
 messagebox.showwarning("предупреджение", "вы не можете использовать этот пароль >:(")  
 return  
 for char in edit\_password\_entry.get():  
 if char == ":" or char == "\*":  
 messagebox.showwarning("предупреждение", "недопустимые символы (:)")  
 return  
 self.transmit(f'editpassword{edit\_password\_entry.get()}')  
 password\_toplevel.destroy()  
  
 password\_toplevel = Toplevel()  
  
 width = password\_toplevel.winfo\_reqwidth()  
 height = password\_toplevel.winfo\_reqheight()  
 x = int((password\_toplevel.winfo\_screenwidth() / 2 - width / 2)+25)  
 y = int((password\_toplevel.winfo\_screenheight() / 2 - height / 2))  
 password\_toplevel.geometry("+{}+{}".format(x, y))  
  
 password\_toplevel.grab\_set()  
 password\_toplevel.resizable(False, False)  
 edit\_password\_label = ttk.Label(password\_toplevel, text="введите новый пароль", anchor="center", justify="center")  
 edit\_password\_entry = ttk.Entry(password\_toplevel)  
 save\_changes\_button = ttk.Button(password\_toplevel, text='сохранить изменения', command=save\_password)  
  
 for i in [edit\_password\_label, edit\_password\_entry, save\_changes\_button]:  
 i.pack()

Вложенная функция edit\_pfp() (Листинг 7.4) позволяет изменить фотографию профиля.

1. Поверх основного окна создается еще одно, содержащее 5 кнопок с картинками. При нажатии на любую из них вызывается вложенная функция choose\_pfp(), в которую передается индекс нажатой кнопки.
2. Функция choose\_pfp() передает на сервер одно сообщение, начинающееся с кода запроса «editpfp», за которым следует индекс выбранного виджета. Переменной profile\_var присваивается выбранная картинка, и окно изменения фотографии профиля закрывается.

Листинг 7.4. Изменение фотографии профиля

def edit\_pfp():  
 def choose\_pfp(pfp\_index):  
 self.transmit(f'editpfp{pfp\_index}')  
 if pfp\_index == '0':  
 self.profile\_var = self.profile\_logo0  
 elif pfp\_index == '1':  
 self.profile\_var = self.profile\_logo1  
 elif pfp\_index == '2':  
 self.profile\_var = self.profile\_logo2  
 elif pfp\_index == '3':  
 self.profile\_var = self.profile\_logo3  
 elif pfp\_index == '4':  
 self.profile\_var = self.profile\_logo4  
 self.profile\_button.configure(image=self.profile\_var)  
 pfps.destroy()  
  
 pfps = Toplevel()  
  
 width = pfps.winfo\_reqwidth()  
 height = pfps.winfo\_reqheight()  
 x = int((pfps.winfo\_screenwidth() / 2 - width / 2)-50)  
 y = int((pfps.winfo\_screenheight() / 2 - height / 2)+50)  
 pfps.geometry("+{}+{}".format(x, y))  
  
 pfps.resizable(False, False)  
 pfps.grab\_set()  
  
 pfp0 = ttk.Button(pfps, image=self.profile\_logo0, command=lambda: choose\_pfp('0'))  
 pfp1 = ttk.Button(pfps, image=self.profile\_logo1, command=lambda: choose\_pfp('1'))  
 pfp2 = ttk.Button(pfps, image=self.profile\_logo2, command=lambda: choose\_pfp('2'))  
 pfp3 = ttk.Button(pfps, image=self.profile\_logo3, command=lambda: choose\_pfp('3'))  
 pfp4 = ttk.Button(pfps, image=self.profile\_logo4, command=lambda: choose\_pfp('4'))  
  
 column = 0  
 for pfp in [pfp0, pfp1, pfp2, pfp3, pfp4]:  
 pfp.grid(row=1, column=column)  
 column += 1

Вложенная функция log\_out() (Листинг 7.5) позволяет выйти из аккаунта. Содержимое файла ‘details.txt’ очищается, главное окно закрывается, и вместо него открывается окно входа в аккаунт.

Листинг 7.5. Выход из аккаунта

def log\_out():  
 with open('details.txt', 'w'):  
 pass  
 self.root.destroy()  
 self.login()

**Со стороны сервера:**

Если сервер получил сообщение, начинающееся с кода запроса “editusername”, то код запроса обрезается, а оставшаяся часть сообщения присваивается переменной username. Далее производится поиск совпадения в базе данных (‘D’ – индекс колонки юзернеймов).

* Если совпадение найдено, клиенту отправляется сообщение «2», означающее, что полученный юзернейм уже найден, и его использовать нельзя.
* Если совпадение не найдено, юзернейм пользователя изменяется в базе данных, и клиенту отправляется сообщение «1», означающее, что юзернейм успешно изменен. Изменения сохраняются.

Листинг 7.6. Обработка кода запроса «editusername»

elif message.startswith('editusername'):  
 username = message[len('editusername'):]  
  
 for cell in sheet['D']:  
 if cell.coordinate != f'D{ID+1}' and username == str(cell.value):  
 transmit(client, 2)  
 print('username already exists')  
 return  
  
 sheet[f'D{ID + 1}'] = username  
 transmit(client, 1)  
 print('username changed')  
 database.save("database.xlsx")

Если сервер получил сообщение, начинающееся с кода запроса «editpassword», то код запроса обрезается, и оставшаяся часть сообщения присваивается переменной password. Пароль пользователя изменяется в базе данных и изменения сохраняются.

Листинг 7.7. Обработка кода запроса «editpassword»

elif message.startswith('editpassword'):  
 password = message[len('editpassword'):]  
 sheet[f'C{ID + 1}'] = password  
 database.save("database.xlsx")  
 print('password changed')

Если сервер получил сообщение, начинающееся с кода запроса «editpfp», то код запроса обрезается, и оставшаяся часть сообщения присваивается переменной index. Индекс фотографии пользователя в базе данных изменяется, изменения сохраняются.

Листинг 7.8. Обработка кода запроса «editpfp»

elif message.startswith('editpfp'):  
 index = message[len('editpfp'):]  
 sheet[f'E{ID+1}'] = index  
 database.save("database.xlsx")  
 print('pfp edited')

# Контакты

**Со стороны клиента:**

Метод contacts\_click() (Листинг 8.1), содержащий вложенные функции find\_friend() (Листинг 8.2), get\_contacts() (Листинг 8.3) и open\_saved() (Листинг 8.4), открывает страницу контактов.

1. Создаются виджеты страницы контактов: лейблы, поле ввода юзернейма, кнопка «найти», по нажатию на которую вызывается вложенная функция find\_friend().
2. Переменная contacts\_click\_check используется для отслеживания текущего состояния страницы контактов. Если она равна False, то это означает, что страница настроек не отображается. В этом случае все виджеты кроме трех основных кнопок (контакты, сообщения и закрыть), скрываются, и размещаются виджеты, принадлежащие странице конактов. Если она равна True, размещаются виджеты главного меню. Это сделано для того, чтобы можно было открывать и закрывать эту страницу по нажатию на одну и ту же кнопку.

Листинг 8.1. Метод contacts\_click(), открывающий страницу контактов

def contacts\_click(self):  
 #find\_friend()  
 #get\_contacts()  
 #open\_saved()  
  
 self.search\_label = ttk.Label(self.root, text='поиск друзей', anchor="center", justify="center", width=35, font=font.Font(size=15))  
 self.seach\_username\_label = ttk.Label(self.root, text='юзернейм')  
 self.search\_entry = ttk.Entry(self.root, width=25)  
 self.search\_button = ttk.Button(self.root, text='найти', command=find\_friend)  
 self.its\_you\_label = ttk.Label(self.root, text='', anchor="center", justify="center")  
  
 self.friend\_username\_label = ttk.Label(self.root, anchor="center", justify="center")  
 self.add\_friend\_button = ttk.Button(self.root, text='отправить сообщение')  
 self.contacts\_label = ttk.Label(self.root, text='мои друзья: ', state='hidden')  
 self.saved\_messages\_button = ttk.Button(self.root, text='открыть избранное', command=open\_saved, width=24)  
  
 if self.contacts\_click\_check:  
 for widget in self.root.winfo\_children():  
 widget.place\_forget()  
 self.contacts\_click\_check = False  
 self.main\_menu\_place()  
 else:  
 self.contacts\_click\_check = True  
 for widget in self.root.winfo\_children():  
 if widget not in [self.contacts\_button, self.messages\_button, self.exit\_button]:  
 widget.place\_forget()  
 get\_contacts()  
 self.search\_label.place(relx=0.5, rely=0.05, anchor="center")  
 self.seach\_username\_label.place(relx=0, rely=0.1)  
 self.search\_entry.place(relx=0.22, rely=0.1)  
 self.search\_button.place(relx=0.75, rely=0.097)  
 self.its\_you\_label.place(relx=0.5, rely=0.21, anchor='center')  
 self.saved\_messages\_button.place(relx=0.25, rely=0.76)  
 self.exit\_button.configure(command=self.main\_menu\_place, text='главное меню')

Вложенная функция find\_friend() (Листинг 8.2) позволяет находить пользователей и открывать с ними чат. Она вызывается по нажатию на кнопку «найти».

1. В начале веденные в поле данные проходят проверку на совпадение с юзернеймом самого пользователя. Если пользователь ввел свой юзернейм, то определенные виджеты скрываются, и выводится сообщение о том, что пользователь нашел сам себя.
2. Если проверка пройдена (пользователь ввел не свой юзернейм), то кнопке add\_friend\_button присваивается функция select\_friend() (это делается тут, а не в основном методе, потому что select\_friend() – это вложенная функция функции find\_friend(), которая, в свою очередь, является вложенной функцией метода contacts\_click(), и доступа к select\_friend() напрямую у него нет).  
   Далее на сервер передается одно сообщение, начинающееся с кода запроса «findfriend», за которым идет юзернейм, который ввел пользователь.
3. Если ответ не получен, производится выход из функции.
4. Если полученный ответ начинается с «1» - такой пользователь существует. На экран выводится текст с юзернеймом этого пользователя, и появляется кнопка «отправить сообщение», по нажатию на которую вызывается функция select\_friend() (Листинг 8.2.1)
5. Если получен ответ «2» - такого пользователя не существует. Лейбл с юзернеймом пользователя и кнопкой «отправить сообщение» скрываются, и выводится соответствующий текст.

Листинг 8.2. Поиск пользователей

def find\_friend():  
 def select\_friend():  
 self.friend\_ID = result[1:]  
 self.friend\_username.set(self.search\_entry.get())  
 self.messages\_click(self.friend\_ID)  
  
 if self.search\_entry.get() == self.username.get():  
 self.its\_you\_label['text'] = 'вы нашли себя!'  
 self.friend\_username\_label.place\_forget()  
 self.add\_friend\_button.place\_forget()  
 else:  
 self.add\_friend\_button.configure(command=select\_friend)  
 self.transmit(f'findfriend{self.search\_entry.get()}')  
 result = self.receive()  
  
 if not result:  
 return  
 if result.startswith('1'):  
 self.its\_you\_label['text'] = 'пользователь найден'  
 self.friend\_username\_label['text'] = self.search\_entry.get()  
 self.friend\_username\_label.place(relx=0.5, rely=0.28, anchor='center')  
 self.add\_friend\_button.place(relx=0.5, rely=0.35, anchor='center')  
 elif result == '2':  
 self.its\_you\_label['text'] = 'такой пользователь не существует >\_<'  
 self.friend\_username\_label.place\_forget()  
 self.add\_friend\_button.place\_forget()

По нажатию на кнопку «отправить сообщение» вызывается функция select\_friend() (Листинг 8.2.1)

Переменным friend\_ID и uname присваиваются значения разбитого по символу «:» ответа, полученного в функции find\_friend() (Листинг 8.2), при этом первый символ ответа обрезается. Переменной friend\_username присваивается значение uname (юзернейм найденного пользователя). Открывается переписка с выбранным пользователем.

Листинг 8.2.1. Отправление сообщения найденному пользователю

def select\_friend():  
 self.friend\_ID, uname = result[1:].split(":")  
 self.friend\_username.set(uname)  
 self.messages\_click(self.friend\_ID)

Вложенная функция get\_contacts() (Листинг 8.3) позволяет получить список контактов пользователя. В ней находятся вложенные функции generate\_page() (Листинг 8.3.1), forward() и back() (Листинг 8.3.2), press() (Листинг 8.3.3).

1. На сервер отправляется код запроса «getcontacts».
2. Если пришел ответ «0», то производится выход из функции – у пользователя нет контактов.
3. В противном случае появляется лейбл с надписью «мои друзья», самый правый символ («\*») полученного ответа (переменная contacts) удаляется, и оставшиеся данные разбиваются по символу «\*».
4. Если полученный список больше 4, то появляются кнопки «🡨» и «🡪», нужные для перелистывания списка контактов. Им присваиваются вложенные функции forward() и back() соответственно (Листинг 8.3.2)
5. Создается пустой список variables, переменной start присваивается значение 0, и запускается функция generate\_page() (Листинг 8.3.1)

Листинг 8.3. Получение списка контактов пользователя

def get\_contacts():

#press(username)  
#generate\_page()  
#forward()  
#back()  
  
 self.start -= 4  
 generate\_page()  
  
 if self.start <= 0:  
 back\_button['state'] = 'disabled'  
 forward\_button['state'] = 'normal'  
  
 self.transmit('getcontacts')  
 contacts = self.receive()  
  
 if contacts == '0':  
 return  
  
 self.contacts\_label.place(relx=0, rely=0.39)  
  
 contacts = contacts.rstrip('\*')  
 contacts = contacts.split('\*')  
  
 if len(contacts) > 4:  
 forward\_button = ttk.Button(self.root, text='->', command=forward)  
 back\_button = ttk.Button(self.root, text='<-', command=back, state='disabled')  
 forward\_button.place(relx=0.75, rely=0.76)  
 back\_button.place(relx=0, rely=0.76)  
  
 self.variables = []  
 self.start = 0  
 generate\_page()

Функция generate\_page() (Листинг 8.3.1) динамически создает кнопки с контактами.

1. Задаются координаты x и y, определяющие, где будет находиться первый контакт.
2. Выполняется цикл for в диапазоне начиная со start и заканчивая минимум из start+4 и длины списка контактов. Вычисление минимума нужно, чтобы не происходила генерация лишних кнопок, если start+4 больше длины списка контактов.
3. В каждой итерации в список variables добавляется строка «contacts», за которой следует i (текущее значение из диапазона).
4. Строка из списка контактов под индексом i разбивается на юзернейм контакта и индекс фотографии контакта по символу «:». В зависимости от полученного индекса переменной img устанавливается соответствующая картинка.
5. Производится динамическое создание переменной и присваивание ей кнопки:

globals() – встроенная функция, возвращающая словарь, который хранит все глобальные переменные. Его ключи – имена переменных.   
Когда мы пишем globals()[self.variables[i]] = ttk.Button(…), мы добавляем в этот словарь ключ “self.variables[i]” и присваиваем ему ttk.Button(…). Таким образом мы создаем переменную с нужным нам именем.

1. При создании кнопки мы присваиваем ей следующую команду:  
   command=lambda i=i: press(globals()[self.variables[i]]['text'])

lambda-функция нужна для того, чтобы можно было передать аргумент в функцию press. Этот аргумент – текст этой самой кнопки.   
Мы обязательно пишем i=i для того, чтобы в функцию press передалось именно текущее значение i. В противном случае значение i у всех кнопок будет последним числом в диапазоне.

1. Кнопки размещаются по заданным x и y, и к значению x прибавляется 0.25, чтобы следующая кнопка разместилась правее.

Листинг 8.3.1. Генерация кнопок с контактами

def generate\_page():  
 x = 0  
 y = 0.45  
 for i in range(self.start, min(self.start+4, len(contacts))):  
 self.variables.append(f'contact{i}')  
 username, index = contacts[i].split(":")  
  
 img = self.profile\_logo0  
 if index == '0':  
 img = self.profile\_logo0  
 elif index == '1':  
 img = self.profile\_logo1  
 elif index == '2':  
 img = self.profile\_logo2  
 elif index == '3':  
 img = self.profile\_logo3  
 elif index == '4':  
 img = self.profile\_logo4  
  
 globals()[self.variables[i]] = ttk.Button(self.root, text=username, image=img, compound='top', command=lambda i=i: press(globals()[self.variables[i]]['text']))  
 globals()[self.variables[i]].place(relx=x, rely=y)  
 x += 0.25

Кнопки 🡨 и 🡪 позволяют перелистывать список контактов. Им присвоены функции forward() (вперед) и back() (назад) соответственно (Листинг 8.3.2)

1. forward() и back() перебирают значения в диапазоне от start до минимума между start+4 и длины списка контактов.
2. Внутри цикла происходит обращение к получаемому вызовом функции globals() словарю, содержащему все глобальные переменные. Мы обращаемся по ключу, который является значением списка variables по

индексу i. variables содержит названия переменных с кнопками.

1. Кнопки, которые эти переменные в себе содержат, скрываются.
2. В случае forward() к значению переменной start прибавляется 4 (переменная start отображает индекс первого контакта на странице. На одной странице отображается 4 контакта, и если мы хотим пролистнуть вперед, должны отобразиться следующие 4 контакта, поэтому прибавляем именно такое число); в случае back() 4, соответственно, отнимается. Вызывается функция generate\_page(), которая генерирует следующую страницу.
3. В случае forward(): кнопка «назад» разблокируется (например, длина списка контактов – 5, мы находимся на первой странице. Кнопка «назад» заблокирована, потому что индекс первого контакта на странице – переменная start – равен 0, нельзя пролистать в минус. Мы нажали «вперед». Кнопка «назад» должна разблокироваться, чтобы была возможность вернуться на предыдущую страницу). Если длина контактов меньше или равна значению start+4, то кнопка «вперед» блокируется. Это сделано для того, чтобы предотвратить генерацию несуществующих контактов.  
   В случае back(): кнопка «вперед» разблокируется (например, длина списка контактов – 5, мы находимся на второй странице. Кнопка «вперед» заблокирована, потому start (индекс первого контакта на странице, в данном случае он равен 4) +4 больше длины списка контактов, поэтому нельзя пролистать вперед. Мы нажали «назад». Кнопка «вперед» должна разблокироваться, чтобы была возможность вернуться на предыдущую страницу). Если значение start меньше или равно 0, кнопка «назад» блокируется, чтобы не листать в минус.

Листинг 8.3.2. Перелистывание списка контактов

def forward():  
 for i in range(self.start, min(self.start+4, len(self.variables))):  
 globals()[self.variables[i]].place\_forget()  
  
 self.start += 4  
 generate\_page()  
  
 if len(contacts) <= self.start+4:  
 forward\_button['state'] = 'disabled'  
 back\_button['state'] = 'normal'  
  
def back():  
 for i in range(self.start, min(self.start + 4, len(self.variables))):  
 globals()[self.variables[i]].place\_forget()  
  
 self.start -= 4  
 generate\_page()  
  
 if self.start <= 0:  
 back\_button['state'] = 'disabled'  
 forward\_button['state'] = 'normal'

Функция press(username) (Листинг 8.3.3) обрабатывает нажатие на выбранную кнопку с контактом.

1. Переменной friend\_username, отвечающей за отображение юзернейма собеседника на странице переписки, присваивается значение переданного аргумента username. Это текст кнопки, на которую нажал пользователь.
2. На сервер отправляется одно сообщение, начинающееся с “getfriendID”, за которым следует переданный аргумент username.
3. Открывается переписка с выбранным пользователем: вызывается метод messages\_click(), в которую передается полученный ID собеседника.

Листинг 8.3.3. Нажатие на контакт

def press(username):  
 self.friend\_username.set(username)  
 self.transmit(f'getfriendID{username}')  
 self.friend\_ID = self.receive()  
 self.messages\_click(self.friend\_ID)

По нажатию на кнопку «открыть избранное» открывается переписка с самим собой: вызывается функция open\_saved() (Листинг 8.4). Переменной friend\_username присваивается значение «избранное», а friend\_ID – None,

Листинг 8.4. Открытие переписки с самим собой

def open\_saved():  
 self.friend\_username.set('избранное')  
 self.friend\_ID = None  
 self.messages\_click(self.friend\_ID)

**Со стороны сервера:**

Если полученное сервером сообщение начинается с кода запроса “findfriend” (Листинг 8.5), переменной username присваивается текст этого сообщения, при этом код запроса обрезается. Далее производится поиск совпадения в базе данных: если совпадение найдено, клиенту отправляется сообщение, содержащее в себе код «1», означающий то, что такой пользователь существует, за которым через знак «:» следуют его ID и юзернейм. Если совпадение не было найдено, отправляется код «2».

Листинг 8.5. Обработка кода запроса «findfriend»

elif message.startswith('findfriend'):  
 username = message[len('findfriend'):]  
 for cell in sheet['D']:  
 if username == str(cell.value):  
 print(f'{username} found')  
 friend\_ID = cell.offset(column=-3).value  
 friend\_username = sheet[f'D{friend\_ID+1}'].value  
 transmit(client, f'1{friend\_ID}:{friend\_username}')  
 return  
 else:  
 print('username not found')  
 transmit(client, '2')

Если сервер получает сообщение, начинающееся с кода запроса «getcontacts», выполняется следующий код:

1. С помощью glob мы получаем все названия txt файлов, записываем их в переменную filenames и создаем пустой список logs.
2. Проходимся циклом по списку filenames, обрезая последние 4 символа – расширения файлов, и добавляем полученные названия в список logs.
3. Создаем пустой список contacts\_ID, и проходимся циклом по списку logs. Если элемент списка равен ID пользователя, отправившего запрос, то ничего не происходит. В противном случае производится попытка разбить элемент списка по знаку ‘-‘ и присвоить полученные значения переменным u1 и u2. Если не получилось, ничего не происходит.
4. Если получилось, проверяем, совпадает ли u1 или u2 с ID пользователя - таким образом мы понимаем, принадлежит ли эта переписка нашему пользователю.
5. Если u1 не равна ID пользователя, добавляем эту переменную в список контактов (это ID того пользователя, с которым общался тот, кто отправил запрос. Соответственно, это его контакт!) В обратном случае добавляем u2.
6. Создаем переменную contacts\_usernames, значение которой – пустая строка. Если длина полученного ранее списка contacts\_ID, отправляем клиенту сообщение «0» - контакты были не найдены.
7. В противном случае проходимся циклом по списку contacts\_ID и ищем в базе данных юзернеймы и индексы фотографий профиля, соответствующие этим ID. К переменной contacts\_usernames каждый раз прибавляем строку, содержащую юзернейм и индекс, разделенные символом «:», и заканчивающуюся символом «\*». Последнее нужно для того, чтобы клиент мог легко разбить итоговую строку, получив в итоге список.
8. Передаем клиенту итоговую строку.

Листинг 8.6. Поиск контактов

elif message.startswith('getcontacts'):  
 filenames = glob.glob('\*.txt')  
 logs = []  
 for file in filenames:  
 file = file[:-4]  
 logs.append(file)  
  
 contacts\_ID = []  
 for log in logs:  
 if log == ID:  
 pass  
 else:  
 try:  
 u1, u2 = log.split('-')  
 if u1 == str(ID) or u2 == str(ID):  
 if u1 != str(ID):  
 contacts\_ID.append(int(u1))  
 else:  
 contacts\_ID.append(int(u2))  
 except:  
 pass  
  
 contacts\_usernames = ""  
 if len(contacts\_ID) == 0:  
 transmit(client, 0)  
 print('no contacts found')  
 return  
 else:  
 for contact in contacts\_ID:  
 username = sheet[f'D{contact+1}'].value  
 pfp = sheet[f'E{contact+1}'].value  
 contacts\_usernames += f'{username}:{pfp}\*'  
 transmit(client, contacts\_usernames)  
 print('contacts sent')

Если полученное сообщение начинается с кода запроса «getfriendID» (Листинг 8.7), код запроса обрезается, и в базе данных ищется нужный ID по переданному юзернейму. Найденный ID отправляется клиенту.

Листинг 8.7. Обработка кода запроса «getfriendID»

elif message.startswith('getfriendID'):  
 username = message[len('getfriendID'):]  
 for cell in sheet['D']:  
 if username == cell.value:  
 friend\_ID = cell.offset(column=-3).value  
 transmit(client, friend\_ID)  
 return

# Сообщения

**Со стороны клиента:**

Вызов метода messages\_click (Листинг 9.1) открывает переписку с выбранным пользователем. В него передается аргумент ID, который обозначает ID пользователя, с которым она будет открыта.

1. Все виджеты скрываются, создается переменная contents, значение которой – пустая строка, и переменная needs\_updating\_check принимает значение True. Она нужна для того, чтобы обозначить, нужно ли в данный момент обновлять историю переписки.
2. Создаются и размещаются необходимые виджеты, задается их стиль.
3. Создается поток update, который запускает функцию load\_messages. Он нужен для того, чтобы обновлять историю переписки, не тормозя основной поток. Этот поток и функция updating() запускаются.

Листинг 9.1. Метод, открывающий переписку

def messages\_click(self, ID):  
 #send\_message(event)  
 #load\_messages()  
 #updating()  
 #clear\_msghistory()  
 #go\_back()

for widget in self.root.winfo\_children():  
 widget.place\_forget()  
  
 self.contents = ''  
 self.needs\_updating\_check = True  
  
 self.back\_button = ttk.Button(self.root, text="<-", command=go\_back, width=3)  
 self.current\_friend = ttk.Label(self.root, textvariable=self.friend\_username, anchor="center", justify="center", font=font.Font(size=12))  
 self.clear\_msghistory\_button = ttk.Button(self.root, text='очистить', width=12, command=clear\_msghistory)  
 self.msghistory = Text(self.root, width=35, height=11, state='disabled', wrap='none')  
 self.text\_widget = Text(self.root, width=37, height=5)  
 self.msghistory['font'] = ('Arial', 11)  
 self.text\_widget['font'] = ('Arial', 11)  
 self.text\_widget.bind('<Return>', send\_message)  
 self.text\_widget.bind('<Key>', self.text\_widget.see('end'))  
  
 self.scrollbar = ttk.Scrollbar(self.root, command=self.msghistory.yview, orient='vertical')  
 self.msghistory.configure(yscrollcommand=self.scrollbar.set)  
  
 self.scrollbar2 = ttk.Scrollbar(self.root, command=self.msghistory.xview, orient='horizontal')  
 self.msghistory.configure(xscrollcommand=self.scrollbar2.set)  
  
 self.msghistory.configure(background='#303030', foreground='#cacaca')  
 self.text\_widget.configure(background='#303030', foreground='#cacaca', insertbackground='#cacaca')  
 self.enter\_message\_label = ttk.Label(self.root, text='начните печатать')  
  
 self.msghistory.place(relx=0, rely=0.1)  
 self.text\_widget.place(relx=0, rely=0.7)  
 self.back\_button.place(x=3, y=3)  
 self.current\_friend.place(relx=0.5, rely=0.05, anchor='center')  
 self.clear\_msghistory\_button.place(relx=0.7, rely=0.01)  
 self.scrollbar.place(relx=0.95, rely=0.1, height=165)  
 self.scrollbar2.place(relx=0, rely=0.65, width=300)  
  
 update = Thread(target=load\_messages)  
 update.start()  
 updating()

Вложенная функция send\_messages (Листинг 9.2) отвечает за отправление сообщений. Она вызывается нажатием на клавишу enter.

1. Проверяется, содержит ли поле ввода сообщений недопустимые символы. Если содержит, выводится предупреждение и сообщение не отправляется.
2. В противном случае мы получаем строку, содержащую текущую дату и время и отправляем на сервер одно сообщение, содержащее код запроса «sendmsg», после которого идет ID собеседника, за которым после знака «\*» идет текущее время, юзернейм отправителя и текст сообщения, разделенные переносом строки.
3. После этого сразу же вызывается функция clear\_everything для того, чтобы удалить перенос строки, появляющийся из-за того, что отправление сообщений происходит по нажатию на клавишу enter.
4. Функция clear\_everything удаляет содержимое поля ввода сообщений.

Листинг 9.2. Отправление сообщений

def send\_message(event):  
 def clear\_everything():  
 self.text\_widget.delete('1.0', 'end')  
  
 for char in self.text\_widget.get('1.0', 'end'):  
 if char == '\*':  
 messagebox.showwarning('предупреждение', 'недопустимые символы (\*)')  
 return  
 current\_time = datetime.datetime.now().strftime("%d.%m.%Y %H:%M:%S")  
 self.transmit(f"sendmsg{ID}\*{current\_time}\n{self.username.get()}:\n{self.text\_widget.get('1.0', 'end')}\n")  
 self.root.after(1, clear\_everything)

Функции load\_messages() и updating() отвечают за обновление истории сообщений (Листинг 9.3). При вызове метода messages\_click() load\_messages() запускается отдельным потоком, а после этого уже в основном потоке запускается updating().

load\_messages():   
пока значение переменной needs\_updating\_check равно True, выполняется цикл, в котором на сервер передается сообщение, начинающееся с кода запроса «loadmsghistory», за которым идет ID собеседника, и с сервера мы получаем содержание переписки. Каждая итерация происходит раз в полсекунды.

updating():

1. Поле истории сообщений становится доступным для редактирования, сохраняются координаты вертикального и горизонтального скроллбаров.
2. В переменную last записывается текущее содержание поля истории сообщений, а в переменную now – содержимое переписки, полученное с сервера. С помощью strip() мы удаляем пробельные символы с начала и с конца.
3. Старое содержимое поля истории сообщений очищается, и вместо него вставляются новые данные.
4. Если значение last равно значению now (обновлений нет), скроллбары остаются на тех же координатах, что и были. Это нужно затем, чтобы дать возможность листать историю, потому что при каждом обновлении содержимого поля истории сообщений оно пролистывается вниз до самого конца. Если обновления есть, поле пролистывается вниз, чтобы пользователь мог увидеть новое сообщение. Поле становится недоступным для редактирования.
5. Если значение переменной self\_updating\_check равно True, через 0.5 секунд эта же функция рекурсивно вызывается заново. Это обеспечивает постоянное обновление истории сообщений.

Листинг 9.3. Обновление истории сообщений

def load\_messages():  
 while self.needs\_updating\_check:  
 self.transmit(f'loadmsghistory{ID}')  
 self.contents = self.receive()  
 time.sleep(0.5)  
  
def updating():  
 try:  
 self.msghistory.configure(state='normal')  
 scroll\_y = self.msghistory.yview()[0]  
 scroll\_x = self.msghistory.xview()[0]  
 last = self.msghistory.get('1.0', 'end').strip()  
 now = self.contents.strip()  
 self.msghistory.delete('1.0', 'end')  
 self.msghistory.insert('1.0', self.contents)  
 if last == now:  
 self.msghistory.yview(MOVETO, scroll\_y)  
 self.msghistory.xview(MOVETO, scroll\_x)  
 else:  
 self.msghistory.see('end')  
 self.msghistory.configure(state='disabled')  
 except Exception:  
 pass  
  
 if self.needs\_updating\_check:  
 self.root.update()  
 self.root.after(ms=500, func=updating)

При нажатии на кнопку «очистить» вызывается функция clear\_msghistory() (Листинг 9.4.) Она очищает историю сообщений у обоих пользователей. На сервер передается сообщение, начинающееся с кода запроса «clearmsghistory», за которым идет ID собеседника,

Листинг 9.4. Очистка истории сообщений

def clear\_msghistory():  
 self.transmit(f'clearmsghistory{ID}')

По нажатию на кнопку «🡨» вызывается функция go\_back(), возвращающая пользователя на главную страницу приложения. Она устанавливает значение переменной needs\_updating\_check на False, что останавливает обновление истории сообщений, и вызывает метод main\_menu\_place().

Листинг 9.5. Возвращение на главную страницу

def go\_back():  
 self.needs\_updating\_check = False  
 self.main\_menu\_place()

Метод main\_menu\_place (Листинг 9.6.) размещает виджеты главной страницы.

Листинг 9.6. Размещение виджетов главной страницы

def main\_menu\_place(self):  
 self.profile\_click\_check = False  
 self.contacts\_click\_check = False  
 self.edit\_username\_check = False  
  
 for widget in self.root.winfo\_children():  
 widget.place\_forget()  
  
 self.profile\_button.place(relx=0, rely=0)  
 self.profile\_label.place(x=0, y=60)  
  
 self.contacts\_button.place(relx=0, rely=0.9)  
 self.messages\_button.place(relx=0.74, rely=0.9)  
 self.exit\_button = ttk.Button(self.root, text='закрыть', command=self.close\_window)  
 self.exit\_button.place(relx=0.5, rely=0.943, anchor='center')  
 self.greeting.place(x=60, y=0)  
 self.help\_button.place(relx=0.75, rely=0)

**Со стороны сервера:**

Если сервер получил сообщение, начинающееся с кода запроса «sendmsg» (Листинг 9.7), выполняется следующий код:

1. Код запроса обрезается, и полученные данные разбиваются по символу «\*». Полученные значения присваиваются переменным friend\_ID и contents.
2. Вложенная функция sendmsg() добавляет в файл с нужной перепиской полученное сообщение.
3. Если значение friend\_ID равно None, это значит, что пользователь отправляет сообщение сам себе. В функцию sendmsg() передается путь файла – ID\_пользователя.txt.
4. В противном случае создаются две переменных: history1 и history2, которым присваиваются строка, содержащая в себе ID пользователя и ID собеседника, разделенные символом ‘-‘. Создается две таких переменных затем, что первый ID в названии файла принадлежит тому, кто первый открыл переписку. Поэтому нужно проверить оба случая. Мы их проверяем и передаем в функцию sendmsg() нужный путь.

Листинг 9.7. Обработка кода запроса «sendmsg»

elif message.startswith('sendmsg'):  
 def sendmsg(path, message):  
 with open(path, 'a') as file:  
 file.write(message)  
 print(f'message sent to {friend\_ID}')  
  
 message = message[len('sendmsg'):]  
 friend\_ID, contents = message.split('\*')  
 if friend\_ID == 'None':  
 sendmsg(f'{ID}.txt', contents)  
 else:  
 history1, history2 = f'{ID}-{friend\_ID}.txt', f'{friend\_ID}-{ID}.txt'  
  
 if os.path.exists(history1):  
 sendmsg(history1, contents)  
 elif os.path.exists(history2):  
 sendmsg(history2, contents)  
 else:  
 sendmsg(history1, contents)

Аналогичным образом в процессе обработки кода запроса «clearmsghistory» (Листинг 9.8) определяется нужный путь, который передается во вложенную функцию clearmsghistory(). Она очищает файл с перепиской.

Листинг 9.8. Обработка кода зароса «clearmsghistory».

elif message.startswith('clearmsghistory'):  
 def clearmsghistory(path):  
 with open(path, 'w'):  
 pass  
 print(f'msghistory with {friend\_ID} cleared')  
  
 friend\_ID = message[len('clearmsghistory'):]  
 if friend\_ID == 'None':  
 clearmsghistory(f'{ID}.txt')  
 else:  
 history1, history2 = f'{ID}-{friend\_ID}.txt', f'{friend\_ID}-{ID}.txt'  
 if os.path.exists(history1):  
 clearmsghistory(history1)  
 elif os.path.exists(history2):  
 clearmsghistory(history2)  
 else:  
 clearmsghistory(history1)

Аналогичным образом в процессе обработки кода запроса «loadmsghistory» определяется нужный путь, который передается во вложенную функцию loadmsghistory(). Она открывает файл в режиме добавления+, ищет самое начало, считывает данные и передает их клиенту.

Листинг 9.9. Обновление истории сообщений

elif message.startswith('loadmsghistory'):  
 def loadmsghistory(path):  
 with open(path, 'a+') as file:  
 file.seek(0)  
 contents = file.read()  
 transmit(client, contents)  
  
 friend\_ID = message[len('loadmsghistory'):]  
  
 if friend\_ID == 'None':  
 loadmsghistory(f'{ID}.txt')  
 else:  
 history1, history2 = f'{ID}-{friend\_ID}.txt', f'{friend\_ID}-{ID}.txt'  
 if os.path.exists(history1):  
 loadmsghistory(history1)  
 elif os.path.exists(history2):  
 loadmsghistory(history2)  
 else:  
 loadmsghistory(history1)

Если была получена неизвестная команда, выполняется следующий код:

else:  
 print('unknown command >\_<')

Заключение

В ходе разработки мессенджера я значительно повысила уровень своих знаний в языке программирования Python: мне пришлось самостоятельно продумывать логику, много читать, находить и анализировать свои ошибки. Это был амбициозный проект, и я даже не была уверена, что у меня действительно получится написать что-то такого уровня, но мотивация работать над этой программой меня не покидала ни разу. Чем больше было трудностей, тем интереснее было разбираться. В будущем я хочу продолжать работать над этим проектом.

В результате моей работы были созданы интерфейсы сервера и клиента, разработаны алгоритмы работы, оформлена документация. Проведенные испытания показали, что программа работает в полном объеме, все требования к программному продукту выполнены.

Источники

<https://docs.python.org/3/library/os.path.html>

<https://docs.python.org/3/library/datetime.html>

<https://docs.python.org/3/library/time.html>

<https://docs.python.org/3/library/socket.html>

<https://docs.python.org/3/library/threading.html>

<https://docs.python.org/3/library/tkinter.html>

<https://docs.python.org/3/library/tkinter.ttk.html>

<https://openpyxl.readthedocs.io/en/stable/>

<https://docs.python.org/3/library/glob.html>

<https://2ip.ru/article/dynvsstatic/>

<https://habr.com/ru/articles/149077/>

<https://thecode.media/meme-tcp/>

<https://www.daniweb.com/programming/software-development/code/487653/access-the-clipboard-via-tkinter>

<https://habr.com/ru/companies/piter/articles/674234/>

<https://treyhunner.com/2020/01/passing-functions-as-arguments/#Lambda_functions_are_an_example_of_this>

<https://www.geeksforgeeks.org/python-globals-function/>