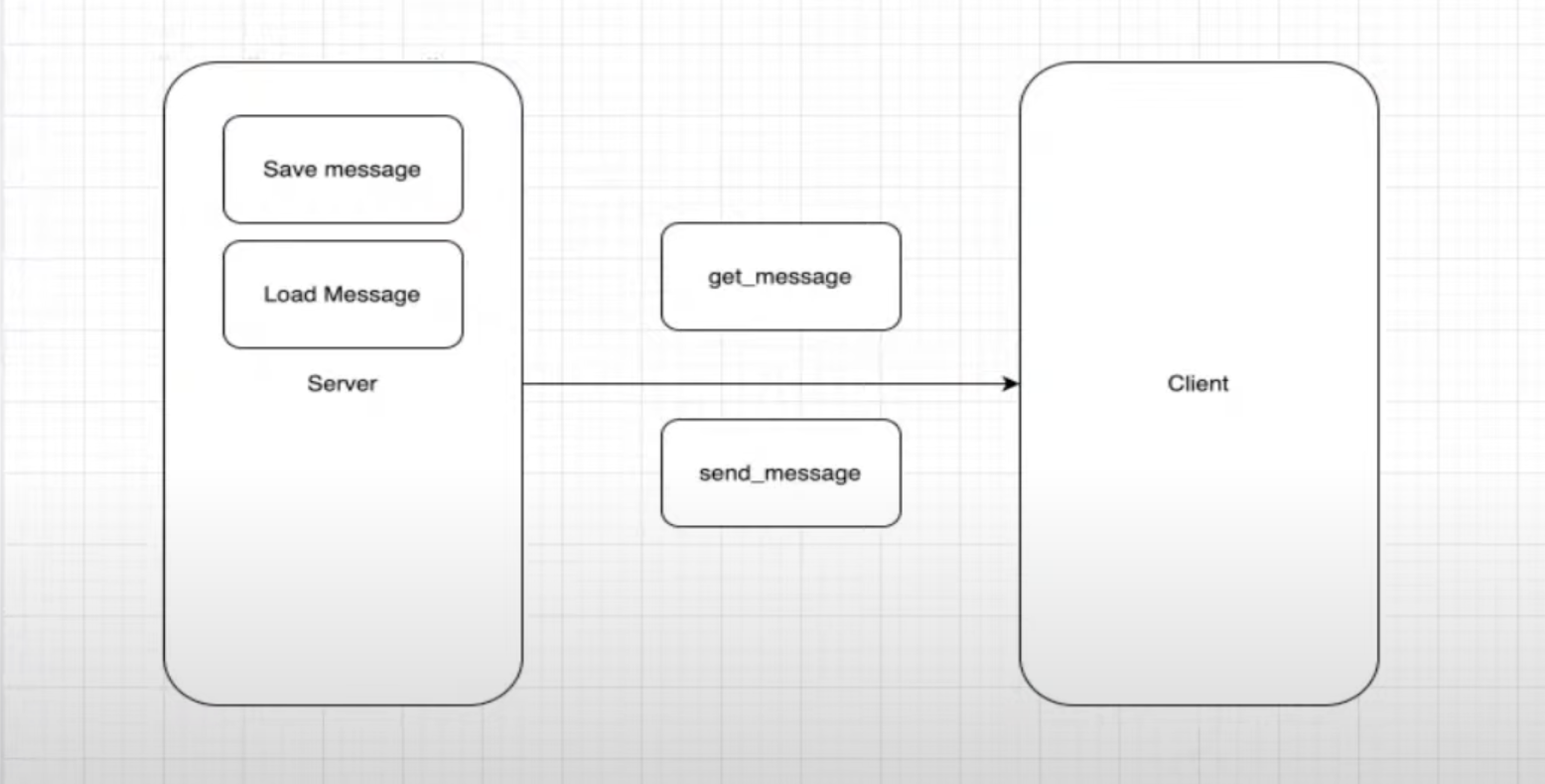
***Практическое задание № 5. Мессенджер на Python***

**Разработка приложения**

Разработаем мессенджер на Python с помощью TCP-протокола, который позволяет получить данные с сервера или отправить их на сервер.

Мессенджер необходимо разрабатывать в среде разработки Python.

Как и в предыдущих лабораторных, мессенджер будет состоять из двух частей: Сервер и Клиент. На Рисунке ниже показано, как работает мессенджер, а также наглядно продемонстрированы функции приложения.



# **Теория**

Вся работа сервера заключается в выполнении двух основных операций:

* Прослушивание клиентских подключений. Если подключается новый клиент, мы добавляем его в нашу коллекцию клиентских сокетов
* Запуск нового потока для каждого подключенного клиента, прием сообщений, отправленных от клиента, и трансляция их всем другим пользователям.

Приведенный ниже код создает TCP-сокет и привязывает его к адресу сервера, а затем прослушивает поступающие соединения:

import socket

from *threading* import Thread

# server's IP address

SERVER\_HOST = "0.0.0.0"

SERVER\_PORT = 5002 # port we want to use

separator\_token = "<SEP>" # we will use this to separate the client name & message

# initialize list/set of all connected client's sockets

client\_sockets = set()

# create a TCP socket

s = socket.socket()

# make the port as reusable port

s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

# bind the socket to the address we specified

s.bind((SERVER\_HOST, SERVER\_PORT))

# listen for upcoming connections

s.listen(5)

print(f"[\*] Listening as {SERVER\_HOST}:{SERVER\_PORT}")

На данном шаге не принимаются соединения, так как не вызывали метод accept(). Приведенный ниже код завершает наш бэкенд:

def listen\_for\_client(cs):

*"""*

*This function keep listening for a message from `cs` socket*

*Whenever a message is received, broadcast it to all other connected clients*

*"""*

while True:

try:

# keep listening for a message from `cs` socket

msg = cs.recv(1024).decode()

except Exception as e:

# client no longer connected

# remove it from the set

print(f"[!] Error: {e}")

client\_sockets.remove(cs)

else:

# if we received a message, replace the <SEP>

# token with ": " for nice printing

msg = msg.replace(separator\_token, ": ")

# iterate over all connected sockets

for client\_socket in client\_sockets:

# and send the message

client\_socket.send(msg.encode())

while True:

# we keep listening for new connections all the time

client\_socket, client\_address = s.accept()

print(f"[+] {client\_address} connected.")

# add the new connected client to connected sockets

client\_sockets.add(client\_socket)

# start a new thread that listens for each client's messages

t = Thread(target=listen\_for\_client, args=(client\_socket,))

# make the thread daemon so it ends whenever the main thread ends

t.daemon = True

# start the thread

t.start()

Как упоминалось ранее, мы добавляем подключенный клиентский сокет в коллекцию наших сокетов. Затем запускаем новый поток и устанавливаем его как поток демона ([daemon thread](https://pythobyte.com/daemon-threads-in-python-c5dbbf24/" \t "_blank)), который выполняет определенную нами функцию listen\_for\_client(). При наличии клиентского сокета эта функция ожидает отправки сообщения с помощью метода recv() и затем отправляет это сообщение всем другим подключенным клиентам.

Для закрытия сокетов используем **close()**:

# close client sockets

for cs in client\_sockets:

cs.close()

# close server socket

s.close()

Клиент выполняет три основные операции:

* Подключение к серверу
* Прослушивание сообщений, поступающих с сервера и вывод их на консоль (чтобы от сервера поступило сообщение, клиент должен отправить его на сервер, а сервер — распространить)
* Ожидание сообщений от пользователей для их дальнейшей отправки на сервер.

Ниже представлен пример подключения к серверу:

import socket

import random

from *threading* import Thread

from *datetime* import datetime

from *colorama* import Fore, init, Back

# init colors

init()

# set the available colors

colors = [Fore.BLUE, Fore.CYAN, Fore.GREEN, Fore.LIGHTBLACK\_EX,

Fore.LIGHTBLUE\_EX, Fore.LIGHTCYAN\_EX, Fore.LIGHTGREEN\_EX,

Fore.LIGHTMAGENTA\_EX, Fore.LIGHTRED\_EX, Fore.LIGHTWHITE\_EX,

Fore.LIGHTYELLOW\_EX, Fore.MAGENTA, Fore.RED, Fore.WHITE, Fore.YELLOW

]

# choose a random color for the client

client\_color = random.choice(colors)

# server's IP address

# if the server is not on this machine,

# put the private (network) IP address (e.g 192.168.1.2)

SERVER\_HOST = "127.0.0.1"

SERVER\_PORT = 5002 # server's port

separator\_token = "<SEP>" # we will use this to separate the client name & message

# initialize TCP socket

s = socket.socket()

print(f"[\*] Connecting to {SERVER\_HOST}:{SERVER\_PORT}...")

# connect to the server

s.connect((SERVER\_HOST, SERVER\_PORT))

print("[+] Connected.")

Далее прослушиваем сообщения с сервера и вывод их на консоль:

def listen\_for\_messages():

while True:

message = s.recv(1024).decode()

print("\n" + message)

# make a thread that listens for messages to this client & print them

t = Thread(target=listen\_for\_messages)

# make the thread daemon so it ends whenever the main thread ends

t.daemon = True

# start the thread

t.start()

Переходим к последней операции — ожиданию сообщений от пользователей с последующей отправкой их на сервер:

while True:

# input message we want to send to the server

to\_send = input()

# a way to exit the program

if to\_send.lower() == 'q':

break

# add the datetime, name & the color of the sender

date\_now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

to\_send = f"{client\_color}[{date\_now}] {name}{separator\_token}{to\_send}{Fore.RESET}"

# finally, send the message

s.send(to\_send.encode())

# close the socket

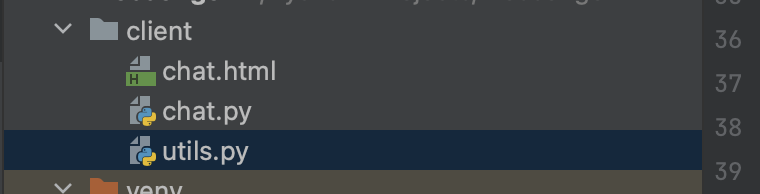
s.close()

# **Начало работы с чатом**

Для последующей работы, предварительно необходимо установить библиотеку flask, позволяющий легко создавать веб-приложения.

После установки библиотеки flask, можно запустить файл main.ру, где уже прописаны настройки приложения с использованием flask.

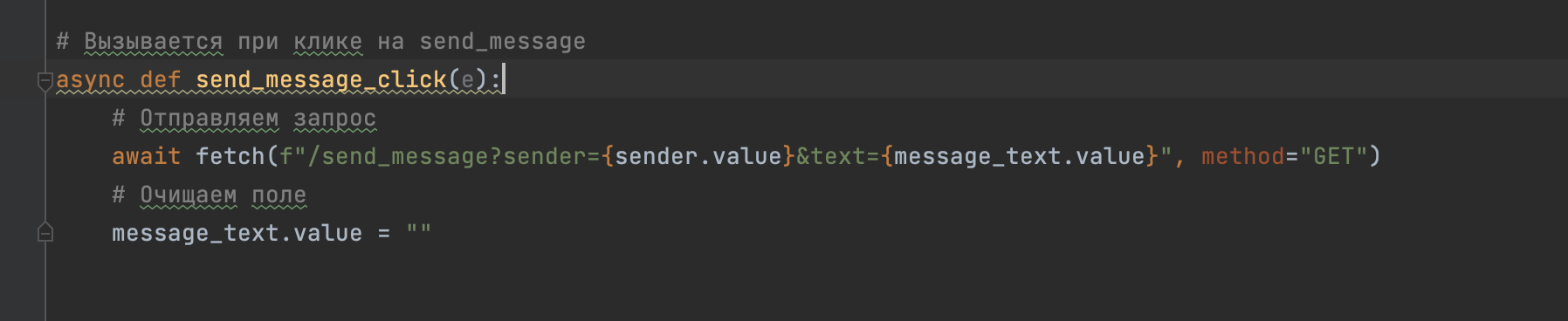
Серверная часть находится в файле main.ру. Клиентская часть находится в папке client, где также расположен файл, отвечающий за интерфейс. Добавляем эту папку в проект с чатом, где располагаются три файла.



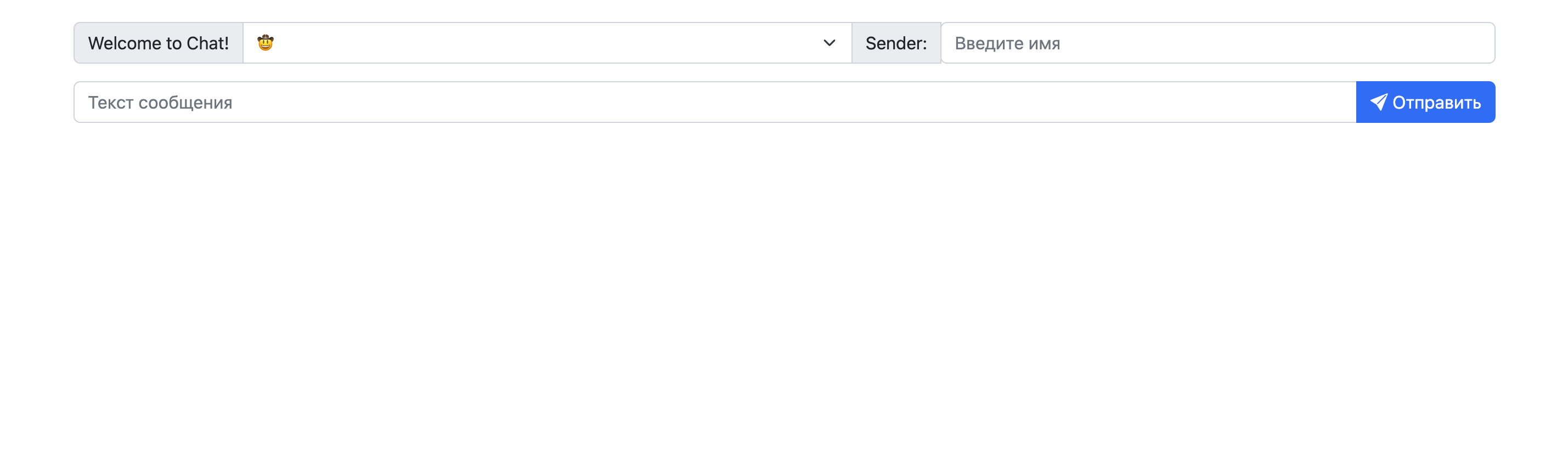
Файл chat.html отвечает за интерфейс чата, интерфейс уже заранее прописан. Код веб-интерфейса приведен ниже.

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="utf-8"/>  
 <meta name="viewport" content="width=device-width,initial-scale=1"/>  
  
 <title>Skull Py Chat</title>  
  
 <link rel="icon" type="image/png" href="favicon.png"/>  
 <link rel="stylesheet" href="https://pyscript.net/latest/pyscript.css"/>  
 <script defer src="https://pyscript.net/latest/pyscript.js"></script>  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" rel="stylesheet"  
 integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous">  
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.9.1/font/bootstrap-icons.css">  
 <py-config>  
 paths = [  
 "/client/utils.py"  
 ]  
 </py-config>  
</head>  
  
<body class="d-flex flex-column min-vh-100">  
  
<py-script src="/client/chat.py"></py-script>  
<div class="container-xl">  
 <div class="input-group my-3 mr-3">  
 <span class="input-group-text">Welcome to Chat!</span>  
  
 <select class="form-select" name="emoji">  
 <option value="🤠">🤠</option>  
 <option value="🥺">🥺</option>  
 <option value="😍">😍</option>  
 <option value="🔥">🔥</option>  
 <option value="🤣">🤣</option>  
 <option value="😭">😭</option>  
 <option value="😂">😂</option>  
 </select>  
  
 <span class="input-group-text">Sender:</span>  
 <div class="col-sm">  
 <input id="sender" type="text" class="form-control" placeholder="Введите имя">  
 </div>  
 </div>  
 <div class="input-group mb-3">  
 <input id="message\_text" name="text" type="text" class="form-control" placeholder="Текст сообщения">  
 <a id="send\_message" class="btn btn-primary">  
 <i class="bi bi-send-fill"></i> Отправить  
 </a>  
 </div>  
 <ul id="chat\_window" class="list-group">  
  
  
 </ul>  
</div>  
</body>  
</html>

Для отправки сообщений используется get-запрос, после отправки сообщения поле с сообщением очищается.



Ниже представлен интерфейс готового чата.



# **Задачи:**

1. Модифицируйте приложение c использованием tcp-сокетов.
2. Переделайте приложение таким образом, чтобы пользователю отображались только новые сообщения, то есть если присоединился новый пользователь, он не должен видеть старые сообщения.
3. Реализовать запись всех сообщений на сервере в файл.
4. Реализовать вывод всех участников чата в форму на интерфейс.
5. Реализовать удаление своих сообщений (пользователь должен иметь возможность удалять только свои сообщения)

Ниже представлен пример модифицированного чата.

