**ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**"ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ"**

Лабораторная работа № 6

Тема: «Изучение протокола обмена сообщениями и данными о присутствии XMPP. Разработка простейшего клиент-серверного приложения "Мессенджер"»

Проверил: Выполнил:

асс. каф. ПИ ст. гр. ПИ-18а

Ищенко А.П. Данильчук К.М.

\_\_\_\_.\_\_\_\_.2021г. \_\_\_\_.\_\_\_\_.2021г.

ст. преп. ПИ

Чернышова А.В.

\_\_\_\_.\_\_\_\_.2021г.

Донецк – 2021

**Задание к лабораторной работе**

Изучить протокол обмена сообщениями и данными о присутствии XMPP, разработать простейшее клиент-серверное программное обеспечение «Мессенджер».

**Краткое описание разработанного протокола**

**Сервер**

Принимает сообщения в формате json, в которых есть параметр type, указывающий на производимую операцию. В зависимости от типа операции меняется количество передаваемых параметров, а также ответ сервера.

**Клиент**

Отправляет json сообщения с производимой операцией и параметрами. А также принимает ответ сервера.

**Скриншоты**

**Сервер**

Таблица клиентов

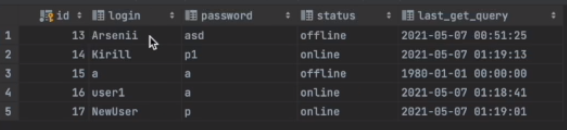
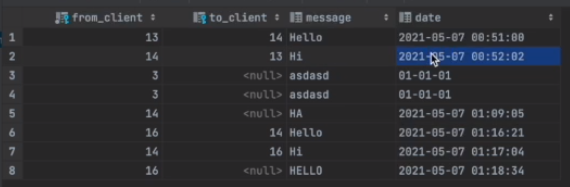


Таблица сообщений



**Клиент**

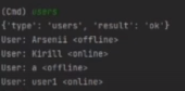
Команда register. Регистрация нового пользователя



Команда login. Вход в систему под существующим аккаунтом



Команда users. Посмотреть информацию о состоянии пользователей



Команда smgs. Отправка сообщения пользователю



Команда sbmsg. Отправка сообщения всем пользователям



Команда msgs. Получение непрочитанных сообщений.



Комада amsgs. Получение всех писем (прочитанных и непрочитанных)



**Программный код**

**------------------------------------Socket.py**

import socket

PACKAGE\_SIZE = 4096 \* 1024

def sendInt(s, value: int) -> None:

s.send(value.to\_bytes(length=4, byteorder='big'))

def getInt(s) -> int:

INT32\_SIZE = 4

result = s.recv(INT32\_SIZE)

result = int.from\_bytes(result, byteorder='big')

return result

def readBytes(s) -> bytearray:

size = getInt(s)

storage = bytearray()

while len(storage) != size:

storage += s.recv(min(PACKAGE\_SIZE, size - len(storage)))

return storage

def writeBytes(s, message\_bytes) -> None:

sendInt(s, len(message\_bytes))

for small\_bytes in [message\_bytes[i - PACKAGE\_SIZE:i] for i in

range(PACKAGE\_SIZE, len(message\_bytes), PACKAGE\_SIZE)]:

s.send(small\_bytes)

s.send(message\_bytes[-(len(message\_bytes) % PACKAGE\_SIZE):])

**------------------------------------XmppClient.py**

import Socket

import socket

import json

def GetStants(client\_socket):

stans = Socket.readBytes(client\_socket)

stans\_dict = json.loads(stans.decode())

return stans\_dict

def SendStants(client\_socket, dict\_data):

dict\_data = json.dumps(dict\_data)

Socket.writeBytes(client\_socket, dict\_data.encode())

class XmppClient:

def \_\_init\_\_(self):

self.client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

def Connect(self, ip="78.31.180.13", port=51000):

self.client\_socket.connect((ip, port))

def Register(self, login, password):

SendStants(self.client\_socket, {"type": "register",

"login": login,

"password": password})

answer = GetStants(self.client\_socket)

print(answer)

def LogIn(self, login, password) -> bool:

SendStants(self.client\_socket, {"type": "login",

"login": login,

"password": password})

answer = GetStants(self.client\_socket)

print(answer)

return answer['result']

def GetUsersList(self):

SendStants(self.client\_socket, {"type": "users"})

users = GetStants(self.client\_socket)

answer = GetStants(self.client\_socket)

print(answer)

return json.loads(users['users'])

def SendMessage(self, to\_client, message):

SendStants(self.client\_socket, {"type": "send",

"to": to\_client,

"message": message})

answer = GetStants(self.client\_socket)

print(answer)

def SendBroadcastMessage(self, message):

self.SendMessage(None, message)

def GetMessages(self, is\_all) -> dict:

SendStants(self.client\_socket, {"type": "get",

"is\_all": is\_all})

messages = GetStants(self.client\_socket)

answer = GetStants(self.client\_socket)

print(answer)

return messages

**------------------------------------XmppServer.py**

import socket

from datetime import datetime

from threading import Thread, Lock

import Socket

import sqlite3

import json

def GetStants(client\_socket):

stans = Socket.readBytes(client\_socket)

stans\_dict = json.loads(stans.decode())

return stans\_dict

def SendStants(client\_socket, dict\_data):

dict\_data = json.dumps(dict\_data)

Socket.writeBytes(client\_socket, dict\_data.encode())

def Now() -> str:

return datetime.now().strftime('%Y-%m-%d %H:%M:%S')

class XmppServer:

def \_\_init\_\_(self):

# Init listen socket

self.server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.server\_socket.bind(("192.168.0.102", 51000))

# self.server\_socket.bind(("localhost", 51001))

self.server\_socket.listen(10)

self.db\_connect = sqlite3.connect("Xmpp/database.sqlite",

check\_same\_thread=False)

self.mutex = Lock()

def Start(self) -> None:

self.db\_connect.execute("update users set status = \'offline\'")

self.db\_connect.commit()

while True:

client\_socket, client\_address = self.Accept()

process = Thread(target=self.UserHandler,

args=[client\_socket, client\_address])

process.start()

def Accept(self):

return self.server\_socket.accept()

def Registration(self, login, password) -> bool:

with self.mutex:

cursor = self.db\_connect.cursor()

cursor.execute(

"select count(\*) from users where login = :login",

{"login": login})

result\_code = False

count\_users\_with\_login = int(cursor.fetchall()[0][0])

if count\_users\_with\_login == 0:

cursor.execute(

"insert into users"

"(login, password, status, last\_get\_query) "

"values(:login, :password, 'online',"

"\'1980-01-01 00:00:00\')",

{"login": login, "password": password})

result\_code = True

self.db\_connect.commit()

return result\_code

def Authentication(self, login, password) -> bool:

with self.mutex:

cursor = self.db\_connect.cursor()

cursor.execute('select password from users where login = :login',

{'login': login})

return password == cursor.fetchone()[0]

def SetStatus(self, login, status):

with self.mutex:

self.db\_connect.execute("update users "

"set status = :status "

"where login = :login",

{'login': login,

'status': status})

self.db\_connect.commit()

def PushMessage(self, from\_client, to\_client, date, message):

with self.mutex:

cursor = self.db\_connect.cursor()

from\_id = cursor.execute("select id from users "

"where login = :login",

{'login': from\_client}).fetchone()[0]

if to\_client is None:

to\_id = None

else:

to\_id = cursor.execute("select id from users "

"where login = :login",

{'login': to\_client}).fetchone()[0]

cursor.execute("insert into messages "

"values (:from\_id, :to\_id, :message, :date)",

{'from\_id': from\_id,

'to\_id': to\_id,

'date': date,

'message': message})

self.db\_connect.commit()

def GetMessage(self, client\_login, all\_message: bool):

with self.mutex:

cursor = self.db\_connect.cursor()

if all\_message:

cursor.execute("select "

"u\_from.login, "

"u\_to.login, "

"message, "

"date "

"from messages "

"join users u\_from on from\_client = u\_from.id "

"left join users u\_to on u\_to.id = to\_client "

"where to\_client is null or "

"u\_to.login = :to\_client or "

"u\_from.login = :from\_client",

{'to\_client': client\_login,

'from\_client': client\_login})

else:

cursor.execute("select last\_get\_query from users "

"where login = :login",

{'login': client\_login})

last\_query\_date = cursor.fetchone()[0]

cursor.execute("select "

"u\_from.login, "

"u\_to.login, "

"message, "

"date "

"from messages "

"join users u\_from on from\_client = u\_from.id "

"left join users u\_to on u\_to.id = to\_client "

"where (to\_client is null or "

"u\_to.login = :to\_client or "

"u\_from.login = :from\_client) "

"and date > :date",

{'to\_client': client\_login,

'from\_client': client\_login,

'date': last\_query\_date})

messages = cursor.fetchall()

cursor.execute("update users "

"set last\_get\_query = :date "

"where login = :login",

{'date': Now(),

'login': client\_login})

self.db\_connect.commit()

return json.dumps(messages)

def GetUsersList(self):

with self.mutex:

cursor = self.db\_connect.cursor()

cursor.execute('select login, status from users')

return json.dumps(cursor.fetchall())

def UserHandler(self, client\_socket, client\_address):

print(f"Start handler for client <{client\_address}>")

user\_login = ''

try:

# State login or register and login

while True:

stans\_dict = GetStants(client\_socket)

type\_message = stans\_dict['type']

while type\_message != 'register' and type\_message != 'login':

SendStants(client\_socket, {

"type": "error",

"text": "Ввойдите в систему"

})

stans\_dict = GetStants(client\_socket)

type\_message = stans\_dict['type']

if type\_message == 'register':

login = stans\_dict['login']

password = stans\_dict['password']

code = self.Registration(login, password)

SendStants(client\_socket, {

"type": "register", "result": code

})

elif type\_message == 'login':

if self.Authentication(stans\_dict['login'],

stans\_dict['password']):

SendStants(client\_socket, {

'type': 'login',

'result': True

})

user\_login = stans\_dict['login']

self.SetStatus(user\_login, 'online')

break

else:

SendStants(client\_socket, {

'type': 'login',

'result': False

})

return

# State run

while True:

stans\_dict = GetStants(client\_socket)

type\_message = stans\_dict['type']

if type\_message == 'send':

self.PushMessage(user\_login, stans\_dict['to'],

Now(), stans\_dict['message'])

elif type\_message == 'get':

messages = self.GetMessage(user\_login,

stans\_dict['is\_all'])

SendStants(client\_socket, {'type': 'messages',

'messages': messages})

elif type\_message == 'users':

users = self.GetUsersList()

SendStants(client\_socket, {

'type': 'users',

'users': users

})

else:

SendStants(client\_socket, {

'type': 'error',

'text': "Комманда недоступна",

'command': type\_message

})

continue

SendStants(client\_socket, {

'type': type\_message,

'result': 'ok'

})

except Exception as e:

print(e)

print(e.args)

print("Exception")

finally:

if user\_login != '':

self.SetStatus(user\_login, 'offline')

client\_socket.close()

**------------------------------------main\_client.py**

import json

import XmppClient

import cmd

# python E:/projects/python/pks\_lab\_6\_xmpp\_client.py

def printMessases(messages):

for message in json.loads(messages['messages']):

print(f"From: {message[0]} To: {message[1]} "

f"message: {message[2]} "

f"date: {message[3]}")

client = XmppClient.XmppClient()

client.Connect(ip="78.31.180.13", port=51001)

def parse(arg):

return tuple(map(str, arg.split()))

class Loginer(cmd.Cmd):

def do\_register(self, arg):

try:

login, password = parse(arg)

client.Register(login, password)

except Exception as e:

print(e)

def do\_login(self, arg):

try:

login, password = parse(arg)

if client.LogIn(login, password):

return True

except Exception as e:

print(e)

class Menu(cmd.Cmd):

def do\_smsg(self, arg):

'Send message'

try:

to, message = parse(arg)

client.SendMessage(to, message)

except Exception as e:

print(e)

def do\_msgs(self, arg):

'Get new messages'

try:

messages = client.GetMessages(False)

printMessases(messages)

except Exception as e:

print(e)

def do\_amsgs(self, arg):

'Get all messages'

try:

messages = client.GetMessages(True)

printMessases(messages)

except Exception as e:

print(e)

def do\_users(self, arg):

'Get users'

try:

users = client.GetUsersList()

for user in users:

print(f"User: {user[0]} <{user[1]}>")

except Exception as e:

print(e)

def do\_sbmsg(self, arg):

'Send broadcast message'

try:

(message,) = parse(arg)

client.SendBroadcastMessage(message)

except Exception as e:

print(e)

loginMenu = Loginer()

mainMenu = Menu()

loginMenu.cmdloop()

mainMenu.cmdloop()

**------------------------------------main\_server.py**

from Xmpp.XmppServer import XmppServer

if \_\_name\_\_ == '\_\_main\_\_':

server = XmppServer()

server.Start()