

ФЕДЕРАЛЬНОЕ АГЕНСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ  
СООБЩЕНИЯ Императора Александра I»

Кафедра «Информационные и вычислительные системы»

Дисциплина «Программирование на языках высокого уровня (Python)»

**ОТЧЁТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9**

Выполнил студент  
Факультет: АИТ  
Группа: ИВБ-211

Шефнер А.

Проверил:

Баталов Д.И.

**Санкт-Петербург**

**2023**

Оценочный лист результатов ЛР № 9

Ф.И.О. студента \_\_\_\_\_ Шефнер Альберт \_\_\_\_\_

Группа \_\_\_\_\_ ИВБ-211 \_\_\_\_\_

№ п/ п	Материалы необходимые для оценки знаний, умений и навыков	Показатель оценивания	Критерии Оценивания	Шкала оценивания	Оценка
1	Лабораторная работа №	Соответствие методике выполнения	Соответствует	7	
			Не соответствует	0	
		Срок выполнения	Выполнена в срок	2	
			Выполнена с опозданием на 2 недели	0	
		оформление	Соответствует требованиям	1 0	
			Не соответствует		
	<b>ИТОГО количество баллов</b>			10	

Доцент кафедры

«Информационные и вычислительные  
системы»

\_\_\_\_\_ 2023 г.

Баталов Д.И. «\_\_»

**1. Напишите программу (клиентскую и серверную часть), позволяющую общаться пользователям внутри локальной сети. Используйте при этом протокол TCP.**

**8. Напишите клиент-серверное приложение, где на клиентской стороне вход в основное окно приложения осуществляется только после его авторизации на сервере, с возможностью регистрации нового пользователя. На серверной стороне хеш связки «логин:пароль» пользователей хранится в отдельном файле.**

### Код

#### Структура проекта

```
albert@RyzenPC ~/u/3/P/Lab 9 (main)> tree . -t
.
├── data
│   └── passwords.json
├── models
│   ├── message.py
│   ├── requests.py
│   └── status.py
├── networking
│   ├── clientside.py
│   ├── common.py
│   ├── password.py
│   └── serverside.py
├── app
│   ├── auth.py
│   ├── clientwindow.py
│   ├── messagebox.py
│   └── chat.py
├── server.py
└── client.py
```

## Общая часть

**networking/common.py** — функции и константы, общие для клиентской и серверной сетевых частей

```
1 import json
2 import socket
3
4 from models.requests import Request
5 from models.status import StatusCode
6
7 HEADER_SIZE = 10
8 ADDRESS = (socket.gethostname(), 5059)
9
10
11
12 def send_text(sock, msg):
13     data = bytes(msg, "utf-8")
14     data += b' ' * (8 - len(data) % 8)
15     header = bytes(f"{len(data):<{HEADER_SIZE}}", "utf-8")
16     sock.send(header + data)
17
18 def send_json_obj(sock, obj):
19     json_ = json.dumps(obj)
20     send_text(sock, json_)
21
22 def send_request(sock, request: Request):
23     send_json_obj(sock, request)
24
25 def respond(sock, code: StatusCode):
26     send_text(sock, str(code))
27
28
29 def recv_text(sock):
30     msg_len = sock.recv(HEADER_SIZE)
31     if len(msg_len) == 0 or msg_len == b'':
32         return None
33
34     msg_len = int(msg_len.decode("utf-8").strip())
35     data = b''
36     while len(data) < msg_len:
37         data += sock.recv(8)
38
39     return data.decode("utf-8").strip()
```

```

✓ 41 def recv_json(sock):
    42     json_ = recv_text(sock)
    43     return json.loads(json_) if json_ is not None else None
    44
✓ 45 def recv_request(sock):
    46     return recv_json(sock)
    47
✓ 48 def recv_response(sock) → StatusCode | None:
    49     code = recv_text(sock)
    50     return StatusCode(int(code)) if code is not None else None

```

**networking/password.py** — функции для работы с логинами и паролями

```

1  import hashlib
2  import json
3  import os
4
5  | PASS_FILE = "data/passwords.json"
6
7  logins = []
8  passwords = []
9
10
✓ 11 def pass_hash(login: str, pass_: str) → str:
12     return hashlib.sha256((pass_ + login).encode()).hexdigest()

```



```

✓ 14 def create_pass_file_if_dont_exists():
✓ 15     if os.path.exists(PASS_FILE):
16         return
17
18     with open(PASS_FILE, 'w') as file:
19         file.write("{}")
20
21 def add_pass(login: str, pass_: str) → None:
22     create_pass_file_if_dont_exists()
23
24     with open(PASS_FILE, 'r') as file:
25         passwords = json.load(file)
26
27     passwords[login] = pass_hash(login, pass_)
28
29     with open(PASS_FILE, 'w') as file:
30         json.dump(passwords, file)
31
32 def check_pass(login: str, pass_: str) → bool:
33     hash_ = pass_hash(login, pass_)
34
35     with open(PASS_FILE, 'r') as file:
36         passwords = json.load(file)
37
38     return login in passwords and passwords[login] == hash_
39
40 def login_exists(login: str):
41     with open(PASS_FILE, 'r') as file:
42         passwords = json.load(file)
43
44     return login in passwords

```

**data/passwords.json** — файл, хранящий хеш-связки логин:пароль

```

1 {
2     "login": "75f49789b5959223c1c875508b5943cddf73b773b62c8ccab4abce7f156de35c",
3     "xd": "84c93d24ac7b174df298c959aad2c72b1c585e72a40a1f82f92715b968694db7",
4     "albert": "21210126833e66a0585cb5aecfa087e68f6c2eace4f852b31af48ab3f0a7a8e9",
5     "vika": "dd3be702dd3aac01a33a69045d924135b4fc90820a489ec41b5b65e0062664a3",
6     "prikol": "b8e9259e0f024d12ac9d0070b9ef4565a2414655e7596932e22bfe37a35c413d",
7     "tester": "30e3524fec2e88cf1f99d5139a9c258ab6bf3f024d0532eda53d5a4ca8af0ba8",
8     "user": "562491275b6806ef82b2cf5db6a4433053811b3e2050f40725d34461c9ee5df7"
9 }

```

**models/message.py** — класс сообщения и нужный для него класс цвета

```
1 import datetime
2 import json
3
4
5 class Color:
6     def __init__(self, r: int, g: int, b: int):
7         if not (0 ≤ r ≤ 255) or \
8             not (0 ≤ g ≤ 255) or \
9             not (0 ≤ b ≤ 255):
10             raise ValueError("Color components must be in range [0, 255]")
11
12         self.r = r
13         self.g = g
14         self.b = b
15
16
17 class Message(dict):
18     TIME_FORMAT = "%Y-%m-%d %H:%M"
19
20     @property
21     def time(self):
22         return datetime.datetime.strptime(self["time"], self.TIME_FORMAT)
23
24     @property
25     def user(self):
26         return self["user"]
27
28     @property
29     def text(self):
30         return self["text"]
31
32     @property
33     def color(self):
34         hashed_user = self.user_hash() % 2**24
35         r = (hashed_user & 0xFF0000) >> 16
36         g = (hashed_user & 0x00FF00) >> 8
37         b = hashed_user & 0x0000FF
38         return Color(r, g, b)
39
40     def __init__(self, user: str, text: str, time: str):
41         super().__init__(user=user, text=text, time=time)
42
43     @staticmethod
44     def new(user: str, text: str) → "Message":
45         time = datetime.datetime.now().strftime(Message.TIME_FORMAT)
46         return Message(user, text, time)
```



```

47
48     @staticmethod
49     def from_json(json_: str):
50         return Message.from_dict(json.loads(json_))
51
52     @staticmethod
53     def from_dict(dct: dict):
54         return Message(dct["user"], dct["text"], dct["time"])
55
56
57     @staticmethod
58     def list_from_json(json_: str):
59         messages = json.loads(json_)
60         return list(map(
61             lambda m: Message.from_dict(m),
62             messages
63         ))
64
65     def __str__(self):
66         return f"[{self.time.hour}:{self.time.minute}] {self.user}: {self.text}"
67
68
69     def user_hash(self):
70         h = 0
71         for c in self["user"]:
72             h = ((h * 7919) ^ (ord(c) * 7907)) * 7901
73         return h % 2**32

```

**models/request.py** — классы и перечисление запросов

```

1 | from enum import StrEnum
2
3
4 | class RequestType(StrEnum):
5     AUTHENTICATION = "authentication"
6     REGISTRATION = "registration"
7     MESSAGE_SEND = "message_send"
8     GET_NEW_MESSAGES = "get_new_messages"
9     CLOSE = "close"
10
11
12 class Request(dict):
13     def __init__(self, type: RequestType, **kwargs):
14         super().__init__(type=type, **kwargs)

```



```

✓ 17 class Authentication(Request):
✓ 18     def __init__(self, login: str, password: str):
19         super().__init__(type=RequestType.AUTHENTICATION, login=login, password=password)
20
21
22 class Registration(Request):
23     def __init__(self, login: str, password: str):
24         super().__init__(type=RequestType.REGISTRATION, login=login, password=password)
25
26
27 class MessageSend(Request):
28     def __init__(self, text: str):
29         super().__init__(type=RequestType.MESSAGE_SEND, text=text)
30
31
32 class GetNewMessages(Request):
33     def __init__(self):
34         super().__init__(type=RequestType.GET_NEW_MESSAGES)
35
36
37 class Close(Request):
38     def __init__(self):
39         super().__init__(type=RequestType.CLOSE)

```

**models/status.py** — перечисление статусных кодов

```

1  from enum import IntEnum
2
3
4  class StatusCode(IntEnum):
5      OK = 200
6      CREATED = 201
7      UNAUTHORIZED = 401
8      FORBIDDEN = 403
9      CONFLICT = 409
10     INTERNAL_ERROR = 502

```

## Серверная часть

**server.py** — отправная точка серверной части

```
1 import socket
2
3 from networking.common import ADDRESS
4 from networking.serverside import ClientSocketThread
5
6 if __name__ == "__main__":
7     clients = {}
8     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9     sock.bind(ADDRESS)
10    sock.listen(5)
11
12    while True:
13        client_sock, address = sock.accept();
14        ClientSocketThread(client_sock, address, clients).start()
```

**networking/serverside.py** — поток работы с клиентами

```
1 import json
2 from queue import Queue
3 from socket import socket
4 from threading import Thread
5 from typing import Dict, Tuple
6
7 from models.message import Message
8 from models.requests import Request, RequestType
9 from models.status import StatusCode
10 from networking.common import recv_request, respond, send_text
11 from networking.password import add_pass, check_pass, login_exists
12
13
14 class ClientSocketThread(Thread):
15     class CloseConnection(Exception):
16         def __init__(self, *args: object) → None:
17             super().__init__(*args)
18
19
20     def __init__(self,
21                 sock: socket,
22                 address: Tuple[str, int],
23                 client_message_queues: Dict[socket, Queue]):
24
25         super().__init__()
26
27         self.sock = sock
28         self.address = address
29         self.client_message_queues = client_message_queues
30         self.authorized = False
31         self.user = None
```

```

33 def run(self) → None:
34     print(f"Connected to {self.address}")
35     self.client_message_queues[self.sock] = Queue()
36
37     while True:
38         try:
39             request = recv_request(self.sock)
40             if request is None:
41                 break
42
43             req_type_str = request["type"].replace('_', ' ').capitalize()
44             print(f"{req_type_str} request from {self.address}")
45
46             self.handle_request(request)
47
48         except self.CloseConnection:
49             break
50         except Exception:
51             print("Internal error while handling request")
52             respond(self.sock, StatusCode.INTERNAL_ERROR)
53
54     self.close()
55
56 def handle_request(self, request: Request):
57     if request["type"] == RequestType.GET_NEW_MESSAGES:
58         self.handle_get_new_messages()
59     elif request["type"] == RequestType.MESSAGE_SEND:
60         self.handle_message_send(request)
61     elif request["type"] == RequestType.AUTHENTICATION:
62         self.handle_authentication(request)
63     elif request["type"] == RequestType.REGISTRATION:
64         self.handle_registration(request)
65     if request["type"] == RequestType.CLOSE:
66         raise self.CloseConnection()
67
68 def handle_get_new_messages(self):
69     messages = []
70     message_queue = self.client_message_queues[self.sock]
71
72     while not message_queue.empty():
73         messages.append(message_queue.get())
74
75     messages_json = json.dumps(messages)
76
77     send_text(self.sock, messages_json)

```



```

78
~ 79     def handle_message_send(self, request):
~ 80         if not self.authorized:
81             respond(self.sock, StatusCode.FORBIDDEN)
82             return
83
84         assert self.user is not None
85
86         message = Message.new(self.user, request["text"])
~ 87         for client in self.client_message_queues.values():
88             client.put(message)
89
90         respond(self.sock, StatusCode.OK)
91
~ 92     def handle_authentication(self, request):
~ 93         if check_pass(request["login"], request["password"]):
94             self.authorized = True
95             self.user = request["login"]
96             respond(self.sock, StatusCode.OK)
97         else:
98             respond(self.sock, StatusCode.UNAUTHORIZED)
99
~ 100     def handle_registration(self, request):
~ 101         if login_exists(request["login"]):
102             respond(self.sock, StatusCode.CONFLICT)
103             return
104
105         add_pass(request["login"], request["password"])
106         respond(self.sock, StatusCode.CREATED)
107         return
108
109
~ 110     def close(self) → None:
111         self.client_message_queues.pop(self.sock)
112         self.sock.close()
113         print(f"Disconnected from {self.address}")

```



## Клиентская часть

**client.py** — отправная точка программы клиента

```
1  import os
2
3  from app.clientwindow import ClientWindow
4  from PySide6.QtWidgets import QApplication
5
6  if __name__ == "__main__":
7      app = QApplication()
8      window = ClientWindow()
9      window.show()
10
11     os._exit(app.exec())
```

**networking/clientside.py** — функции клиентской сетевой части

```
1  from socket import socket
2  from typing import List
3
4  from models.message import Message
5  from models.requests import (Authentication, Close, GetNewMessages,
6                               MessageSend, Registration)
7  from models.status import StatusCode
8  from networking.common import recv_response, recv_text, send_request
9
10
11  def authorize(sock: socket, login: str, password: str) → StatusCode | None:
12      send_request(sock, Authentication(login, password))
13      return recv_response(sock)
14
15
16  def register(sock: socket, login: str, password: str) → StatusCode | None:
17      send_request(sock, Registration(login, password))
18      return recv_response(sock)
19
20
21  def send_message(sock: socket, text: str) → StatusCode | None:
22      send_request(sock, MessageSend(text))
23      return recv_response(sock)
```

```

25
✓ 26 def get_new_messages(sock: socket) → List[Message] | None:
27     send_request(sock, GetNewMessages())
28     messages_json = recv_text(sock)
✓ 29     if messages_json is None:
30         return None
31
32     return Message.list_from_json(messages_json)
33
✓ 34 def close_req(sock):
35     send_request(sock, Close())

```

**app/clientwindow.py** — окно клиентского приложения

```

1 import socket
2
3 from app.auth import AuthWidget, RegWidget
4 from app.chat import ChatWidget
5 from app.messagebox import info, warning
6 from models.status import StatusCode
7 from networking.clientside import authorize, register
8 from networking.common import ADDRESS
9 from PySide6.QtCore import Slot
10 from PySide6.QtWidgets import QMainWindow
11
12
13 class ClientWindow(QMainWindow):
14     def __init__(self) → None:
15         super().__init__()
16         sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17         sock.connect(ADDRESS)
18
19         self.sock = sock;
20
21         self.setGeometry(0, 0, 300, 450)
22         self.setWindowTitle("Chat")
23         self.switch_to_auth()

```



```

25     @Slot()
26     def switch_to_auth(self):
27         auth_widget = AuthWidget()
28         auth_widget.reg_button.clicked.connect(self.switch_to_reg)
29         auth_widget.enter_button.clicked.connect(self.auth)
30
31         self.setCentralWidget(auth_widget)
32
33     @Slot()
34     def switch_to_reg(self):
35         reg_widget = RegWidget()
36         reg_widget.reg_button.clicked.connect(self.reg)
37         reg_widget.back_button.clicked.connect(self.switch_to_auth)
38         self.setCentralWidget(reg_widget)
39
40     @Slot()
41     def switch_to_chat(self):
42         chat_widget = ChatWidget(self.sock)
43         self.setCentralWidget(chat_widget)
44
45     @Slot()
46     def auth(self):
47         auth_widget = self.centralWidget()
48         assert isinstance(auth_widget, AuthWidget)
49
50         login = auth_widget.login_edit.text()
51         password = auth_widget.pass_edit.text()
52
53         if login == "" or password == "":
54             warning(self, "Логин или пароль не должны быть пустыми.")
55             return
56
57         result = authorize(self.sock, login, password)
58         if result == StatusCode.OK:
59             print("Successfully logged in.")
60             self.setWindowTitle("Chat - " + login)
61             self.switch_to_chat()
62         else:
63             warning(self, "Неправильный логин или пароль! Попробуйте снова...")
64
65     @Slot()
66     def reg(self):
67         reg_widget = self.centralWidget()
68         assert isinstance(reg_widget, RegWidget)
69
70         login = reg_widget.login_edit.text()
71         password = reg_widget.pass_edit.text()
72         confirm = reg_widget.confirm_edit.text()

```

```

73
74     if login == '':
75         warning(self, "Логин не должен быть пустым.")
76         return
77
78     if password == '':
79         warning(self, "Пароль не должен быть пустым")
80         return
81
82     if password != confirm:
83         warning(self, "Пароли не совпадают!\nПовторите ввод.")
84         return
85
86
87     result = register(self.sock, login, password)
88     if result == StatusCode.CONFLICT:
89         warning(self, "Данный логин уже существует.\nПопробуйте другой")
90         return
91
92     elif result == StatusCode.CREATED:
93         info(self,
94             "Вы успешно зарегистрированы!\nИспользуйте ваш новый логин для входа."
95         )
96     self.switch_to_auth()

```

**app/messagebox.py** — вспомогательные функции для вызова маленьких окон с сообщением

```

1  from PySide6.QtWidgets import QMessageBox
2
3
4  def warning(self, text: str) → None:
5      QMessageBox().warning(
6          self, "",
7          text,
8          QMessageBox.Ok, # type: ignore
9          QMessageBox.Ok # type: ignore
10     )
11
12  def info(parent, text: str) → None:
13      QMessageBox().information(
14          parent, "",
15          text,
16          QMessageBox.Ok, # type: ignore
17          QMessageBox.Ok # type: ignore
18     )

```



## app/auth.py — виджеты авторизации и регистрации

```
1 | from PySide6.QtCore import Qt
2 | from PySide6.QtGui import QFont
3 | from PySide6.QtWidgets import (QHBoxLayout, QLabel, QLineEdit, QPushButton,
4 |                               QVBoxLayout, QWidget)
5 |
6 | FONT_FAMILY = "Noto Sans"
7 |
8 | class AuthWidget(QWidget):
9 |     def __init__(self) → None:
10 |         super().__init__()
11 |
12 |         primary_font = QFont(FONT_FAMILY, 32, QFont.Bold) # type: ignore
13 |         secondary_font = QFont(FONT_FAMILY, 20)
14 |
15 |         self.auth_label = QLabel("Авторизация")
16 |         self.auth_label.setFont(primary_font)
17 |
18 |         self.login_label = QLabel("Логин:")
19 |         self.login_label.setFont(secondary_font)
20 |         self.login_edit = QLineEdit()
21 |
22 |         self.pass_label = QLabel("Пароль:")
23 |         self.pass_label.setFont(secondary_font)
24 |         self.pass_edit = QLineEdit()
25 |
26 |         self.enter_button = QPushButton("Войти")
27 |         self.enter_button.setFont(secondary_font)
28 |
29 |         self.reg_button = QPushButton("Зарегистрироваться")
30 |         self.reg_button.setFont(secondary_font)
31 |
32 |         buttons_layout = QHBoxLayout()
33 |         buttons_layout.addWidget(self.enter_button)
34 |         buttons_layout.addWidget(self.reg_button)
35 |
36 |         vbox = QVBoxLayout()
37 |         vbox.addWidget(self.auth_label)
38 |         vbox.addStretch(1)
39 |         vbox.addWidget(self.login_label)
40 |         vbox.addWidget(self.login_edit)
41 |         vbox.addStretch(1)
42 |         vbox.addWidget(self.pass_label)
43 |         vbox.addWidget(self.pass_edit)
44 |         vbox.addStretch(1)
45 |         vbox.addLayout(buttons_layout)
46 |         vbox.addSpacing(30)
47 |         vbox.setAlignment(Qt.AlignJustify) # type: ignore
48 |
```

```
49     hbox = QHBoxLayout()
50     hbox.addSpacing(30)
51     hbox.addLayout(vbox)
52     hbox.addSpacing(30)
53     self.setLayout(hbox)
54
55
56 class RegWidget(QWidget):
57     def __init__(self) → None:
58         super().__init__()
59
60         primary_font = QFont(FONT_FAMILY, 32, QFont.Bold) # type: ignore
61         secondary_font = QFont(FONT_FAMILY, 20)
62
63         self.reg_label = QLabel("Регистрация")
64         self.reg_label.setFont(primary_font)
65
66         self.back_button = QPushButton("Назад")
67         self.back_button.setFont(secondary_font)
68
69         self.login_label = QLabel("Логин:")
70         self.login_label.setFont(secondary_font)
71         self.login_edit = QLineEdit()
72
73         self.pass_label = QLabel("Пароль:")
74         self.pass_label.setFont(secondary_font)
75         self.pass_edit = QLineEdit()
76
77         self.confirm_label = QLabel("Подтвердите пароль:")
78         self.confirm_label.setFont(secondary_font)
79         self.confirm_edit = QLineEdit()
80
81
82         self.reg_button = QPushButton("Зарегистрироваться")
83         self.reg_button.setFont(secondary_font)
84
85         top_hbox = QHBoxLayout()
86         top_hbox.addWidget(self.back_button)
87         top_hbox.addWidget(self.reg_label)
88
89         vbox = QVBoxLayout()
90         vbox.addLayout(top_hbox)
91         vbox.addStretch(1)
92         vbox.addWidget(self.login_label)
93         vbox.addWidget(self.login_edit)
94         vbox.addStretch(1)
95         vbox.addWidget(self.pass_label)
96         vbox.addWidget(self.pass_edit)
97         vbox.addStretch(1)
98         vbox.addWidget(self.confirm_label)
```



```

99         vbox.addWidget(self.confirm_edit)
100        vbox.addStretch(1)
101        vbox.addWidget(self.reg_button)
102        vbox.addSpacing(30)
103        vbox.setAlignment(Qt.AlignJustify) # type: ignore
104
105        hbox = QHBoxLayout()
106        hbox.addSpacing(30)
107        hbox.addLayout(vbox)
108        hbox.addSpacing(30)
109        self.setLayout(hbox)

```

### app/chat.py — виджет чата

```

1 from app.messagebox import warning
2 from models.message import Message
3 from models.status import StatusCode
4 from networking.clientside import get_new_messages, send_message
5 from PySide6.QtCore import Qt, QTimer, Slot
6 from PySide6.QtGui import QFont
7 from PySide6.QtWidgets import (QHBoxLayout, QLineEdit, QPushButton, QTextEdit,
8                                QVBoxLayout, QWidget)
9
10 FONT_FAMILY = "Noto Sans"
11
12 class MessageEnterWidget(QWidget):
13     def __init__(self, sock, timer):
14         super().__init__()
15
16         self.sock = sock
17         self.timer = timer
18
19         self.init_ui()
20
21         self.send_button.clicked.connect(self.send_message)
22
23     def init_ui(self):
24         self.line_edit = QLineEdit()
25         self.line_edit.setFont(QFont(FONT_FAMILY, 14))
26
27         self.send_button = QPushButton()
28         self.send_button.setFont(QFont(FONT_FAMILY, 14))
29
30         self.send_button.setText("Отправить")
31         self.send_button.setFont(QFont(FONT_FAMILY, 14))
32
33         layout = QHBoxLayout()
34         layout.addWidget(self.line_edit)
35         layout.addWidget(self.send_button)
36         self.setLayout(layout)
37

```

```

40 @Slot()
41 def send_message(self):
42     message = self.line_edit.text().strip()
43     self.line_edit.clear()
44
45     if len(message) == 0:
46         return
47
48     response = send_message(self.sock, message)
49     if response is not StatusCode.OK:
50         self.timer.stop()
51         warning(self, "Ошибка при отправке сообщения")
52
53
54 class ChatWidget(QWidget):
55     def __init__(self, sock):
56         super().__init__()
57
58         self.sock = sock
59
60         self.timer = QTimer(self)
61         self.timer.timeout.connect(self.update_messages)
62         self.timer.start(300)
63
64         self.init_ui()
65
66
67     def init_ui(self):
68         layout = QVBoxLayout()
69
70         self.messages_area = QTextEdit("<html></html>")
71         self.messages_area.setAlignment(Qt.AlignBottom) # type: ignore
72         self.messages_area.setReadOnly(True)
73         self.messages_area.ensureCursorVisible()
74         self.messages_area.setFont(QFont(FONT_FAMILY, 14))
75
76         self.message_enter_field = MessageEnterWidget(self.sock, self.timer)
77         self.message_enter_field.setFont(QFont(FONT_FAMILY, 14))
78
79         layout.addWidget(self.messages_area)
80         layout.addWidget(self.message_enter_field)
81
82         self.setLayout(layout)
83

```



```

85     @Slot()
86     def update_messages(self):
87         messages = get_new_messages(self.sock)
88         if messages is None:
89             warning(self, "Ошибка при получении сообщений")
90             return
91
92         for message in messages:
93             self.append_message(message)
94
95
96     def append_message(self, message: Message):
97         r = format(message.color.r, '02x')
98         g = format(message.color.g, '02x')
99         b = format(message.color.b, '02x')
100        msg_html = f'[{message.time.hour}:{message.time.minute}] ' + \
101            f'<span style="color:#{r}{g}{b};">{message.user}</span> ' + \
102            message.text
103        self.messages_area.append(msg_html)

```

## Отладка

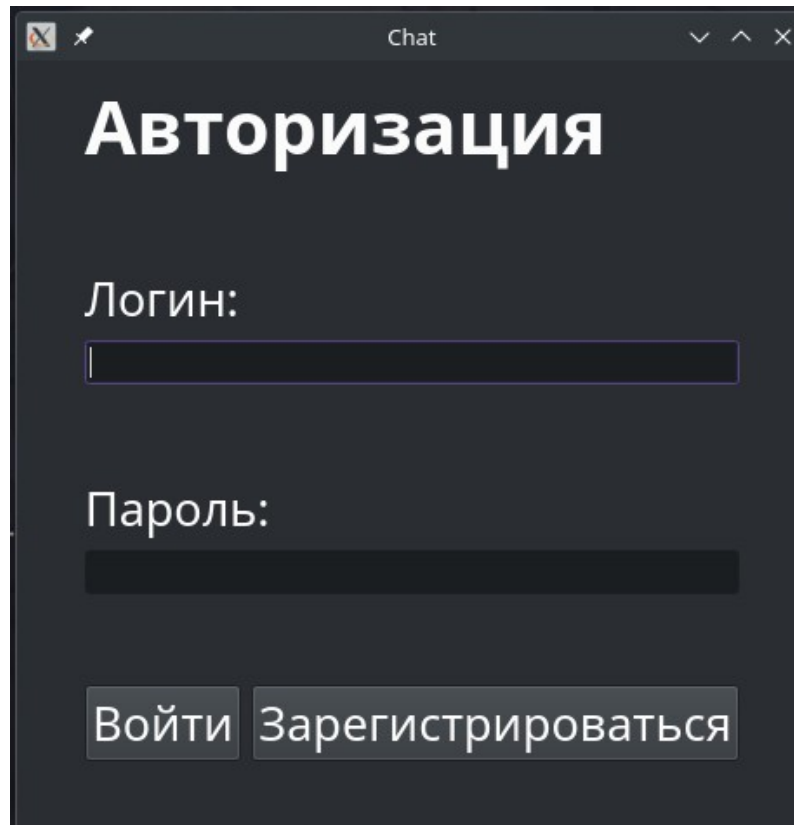
Логи сервера, которые выводятся в консоль

```

Connected to ('172.17.0.1', 39938)
Authentication request from ('172.17.0.1', 39938)
Get new messages request from ('172.17.0.1', 39938)
Get new messages request from ('172.17.0.1', 39938)
Get new messages request from ('172.17.0.1', 39938)
Get new messages request from ('172.17.0.1', 39938)
Get new messages request from ('172.17.0.1', 39938)
Message send request from ('172.17.0.1', 39938)
Get new messages request from ('172.17.0.1', 39938)
Get new messages request from ('172.17.0.1', 39938)
Get new messages request from ('172.17.0.1', 39938)
Get new messages request from ('172.17.0.1', 39938)
Disconnected from ('172.17.0.1', 39938)

```

## Клиентская часть — окно авторизации



The screenshot shows a window titled "Chat" with a dark background. At the top, the title bar contains a close button, a maximize button, and the text "Chat". Below the title bar, the word "Авторизация" (Authorization) is displayed in a large, bold, white font. Underneath, the label "Логин:" (Login:) is followed by an empty text input field. Below that, the label "Пароль:" (Password:) is followed by a password input field. At the bottom, there are two buttons: "Войти" (Login) and "Зарегистрироваться" (Register).

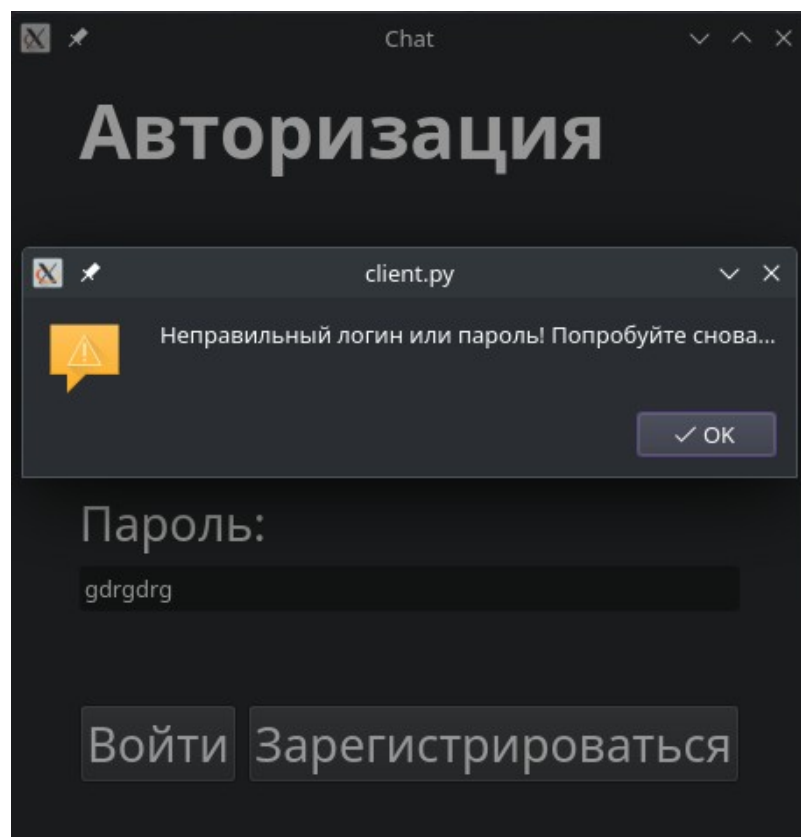
Chat

# Авторизация

Логин:

Пароль:

Войти Зарегистрироваться



This screenshot shows the same authorization window as the previous one, but with an error message displayed. The error message is in a small dialog box with a yellow warning icon and the text "Неправильный логин или пароль! Попробуйте снова..." (Incorrect login or password! Try again...). Below the error message, the password input field now contains the text "gdrgrgrg". The "Войти" (Login) and "Зарегистрироваться" (Register) buttons remain at the bottom.

Chat

# Авторизация

Неправильный логин или пароль! Попробуйте снова...

Пароль:

Войти Зарегистрироваться

