

Правительство Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Кафедра «Компьютерная безопасность»

Отчет
к лабораторной работе №6
по дисциплине
«Языки программирования»

Работу выполнил
студент группы СКБ-201

А.Н. Ушаков

подпись, дата

Работу проверила

М.Ю. Монина

подпись, дата

Москва 2021

Содержание

ПОСТАНОВКА ЗАДАЧИ	3
1. АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ.....	6
1.1 Приложение XML Reader.....	6
1.2 Приложение Server.....	7
1.2.1 Класс Server_Settings	7
1.2.2 Класс Server_Settings	8
1.2.3 Класс Server_Settings	9
1.3 Приложение Client.....	10
1.3.1 Класс DialogAboutMe	11
1.3.2 Класс Full_Size	12
1.3.3 Класс User_Name	12
1.3.4 Класс Color_Message.....	13
1.3.5 Класс Send_User.....	13
1.3.6 Класс Server_Settings	14
1.3.7 Класс Status_User.....	14
1.3.8 Класс User_Info	14
1.3.9 Класс User.....	15
1.3.10 Класс MainWindow	17
2. ВЫПОЛНЕНИЕ ЗАДАЧ	19
2.1 Приложение XML Reader.....	19
2.2 Приложение Server.....	19
2.3 Приложение Client.....	21
3. ПОЛУЧЕНИЕ ИСПОЛНЯЕМЫХ МОДУЛЕЙ.....	25
4. ТЕСТИРОВАНИЕ	25
ПРИЛОЖЕНИЕ А	46
ПРИЛОЖЕНИЕ Б	53
ПРИЛОЖЕНИЕ В	75

Постановка задачи

Разработать программу на языке Си++ (ISO/IEC 14882:2014), демонстрирующую решение поставленной задачи.

Общая часть

Тема: Сетевое взаимодействие. Поддержка сети, потоки, работа с XML.

Задание на 8 баллов.

Разработать графическое приложение с использованием библиотеки Qt – сетевой чат, содержащее клиентскую и серверную часть.

Сервер. Заголовок окна сервера должен содержать ip-адрес и порт, на котором запущен сервер, а также количество подключенных клиентов.

Окно сервера содержит информацию о подключении/отключении клиентов (журнал), а также главное меню состоящее из пунктов:

1) Файл [кнопки]

- a.** Включить
- б.** Выключить
- в.** Сохранить журнал в XML-файл
- г.** Выход

2) Настройки [кнопки]

а. Сеть [кнопка] - при нажатии открывается модальное диалоговое окно для ввода значений ip-адреса и порта (*по умолчанию 127.0.0.1 и 45678*).

Клиент. Заголовок окна клиента должен содержать ip-адрес и порт *сервера к которому подключен*, а также статус пользователя.

Окно клиента (наследуется от QMainWindow) содержит главное меню состоящее из пунктов:

3) Файл [кнопки]

- а.** Подключиться к «*серверу*» (задается в меню **Настройки**)
- б.** Отключится
- в.** Сохранить историю в XML-файл
- г.** Выход

4) Настройки

а. Сервер: ip-адрес и порт сервера [кнопка] - при нажатии открывается модальное диалоговое окно для ввода значений (*по умолчанию 127.0.0.1 45678*), после ввода значений текст кнопки меняется.

б. Вкл/Выкл автоматический прием файлов [галочка]

- в.** Имя пользователя [кнопка] - позволяет изменить имя пользователя
- г.** Фото пользователя [кнопка] - позволяет изменить изображение пользователя
- д.** Статус [дочернее меню] - варианты
- i)** доступен
- ii)** отошел
- iii)** не беспокоить
- iv)** *другой* - открывает меню изменения текста, заменяет надпись кнопки, отображение на кнопке 16 первых символов + троеточие.

5) Вид

- а.** Выбор цвета фона [кнопка] - открывает модальное окно выбора цвета
- б.** Выбор цвета сообщений [кнопка] - открывает модальное окно выбора цвета информации об отправителе и *[базового]* цвета сообщения
- в.** Вкл/Выкл отображения строки состояния [галочка]
- г.** Вкл/Выкл отображения ip-адреса отправителя [галочка]
- д.** Вкл/Выкл отображения времени сообщения [галочка]

6) Справка

- а.** О программе - открывает модальное окно, содержащее фото и имя автора, дату сборки, версию Qt с которой собиралось, версию Qt с которой запущено, кнопку, закрывающую окно

В центральной части окна располагается область сообщений (*рекомендуется использовать QRichText*), под которой строка сообщения и мультикнопка отправки (по умолчанию оправка, выпадающие варианты: отправить картинку - открывает модальный диалог выбора файла картинки; отправка форматированного сообщения - изменяет внешний вид строки сообщения, добавляя к ней кнопки выбора шрифта и выбора цвета), после нажатия фиксируется *по умолчанию*. Справа от центральной части располагается список подключенных пользователей, состоящий из уменьшенного фото после которого стоит имя пользователя.

При нажатии правой кнопкой на (отправленной) картинке в области сообщений появляется контекстное меню: открыть в полном размере - создающее новое (немодальное) окно в котором отображается изображение в масштабе 1:1; сохранить изображение - позволяющее сохранить изображение в файл.

При нажатии правой кнопкой в списке пользователей на имени пользователя появляется контекстное меню: информация - создающая модальное окно с ip-адресом, (датой и) временем подключения, статусом пользователя; отправить файл - создающая немодальное окно с

кнопкой открытия файла для отправки и прогрессом отправки.

При поступлении сообщения (получаемые картинки масштабируются до 320*240), если текущий статус отличен от “**не беспокоить**”, раздается звук.

При получении файла, если **автоматический прием включен**, в область сообщений заносится информация вида “от кого + имя файла + размер” (аналогичная обычным сообщениям). Если **автоматический прием выключен**, открывается немодальное окно с подтверждением приема файла - при подтверждении в нем отображается прогресс передачи. Если текущий статус отличен от “**не беспокоить**”, раздается звук (в обоих случаях).

Если “принимаемый” файл существует - создается файл к имени которого добавляется цифра. Нижнюю часть окна занимает строка состояния. Информация разделена на *два* столбца: количество принимаемых файлов, общий прогресс, размер в килобайтах/мегабайтах/...; количество передаваемых файлов, общий прогресс, размер в килобайтах/мегабайтах/... .

Приложение должно сохранять свои настройки (*в ini-файл, использовать QSettings*)

При нажатии “Сохранить историю в XML-файл” (в меню Файл) вывести диалог открытия файла после чего сформировать XML-файл содержащий полную информацию из истории переписки:

- 1)** дату и время поступления сообщения
- 2)** ip-адрес отправителя
- 3)** имя отправителя
- 4)** тело сообщение [варианты]
 - a)** если простое - сохранить текст
 - b)** если с форматированием - сохранить форматирование и текст
 - c)** если картинка - сохранить масштабированную версию (*в Base64*)
 - d)** если файл - сохранить имя файла и его MD5-хэш

Задание на +2 балла.

Разработать приложение для просмотра истории переписки из XML файла.

1. Алгоритм решения задачи

Для решения поставленных задач были разработаны приложения: XML Reader, Server, Client.

1.1 Приложение XML Reader

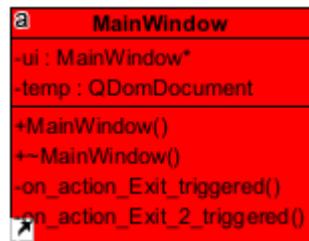


Рис. 1: UML-диаграмма класса MainWindow

1.1.1 Конструктор

- 1) MainWindow()

1.1.2 Методы слоты класса

- 1) on_action_Exit_triggered()
- 2) on_action_Exit_2_triggered()

1.1.3 Деструктор

- 1) ~MainWindow()

1.2 Приложение Server

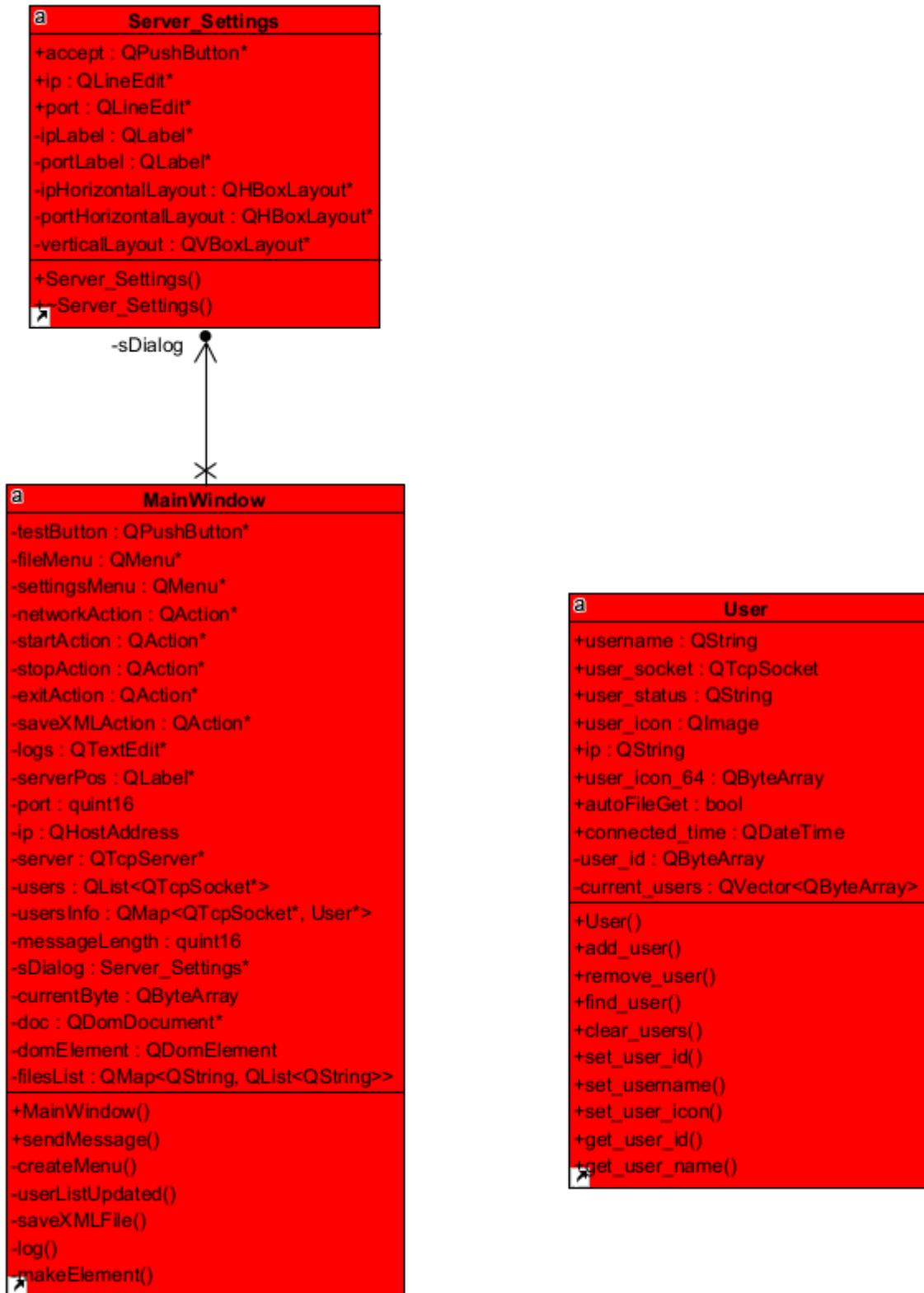


Рис. 2: UML-диаграмма приложения Server

1.2.1 Класс Server_Settings

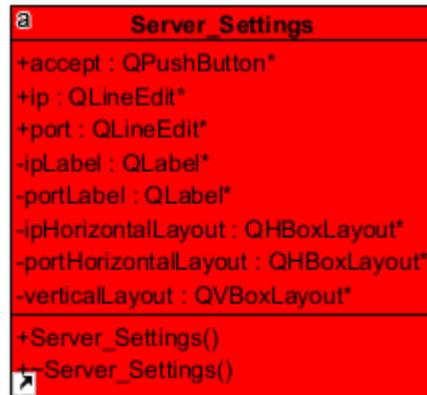


Рис. 3: UML-диаграмма класса Server_Settings

1.2.1.1 Конструктор

1) Server_Settings()

1.2.1.2 Деструктор

1) ~Server_Settings()

1.2.2 Класс Server_Settings



Рис. 4: UML-диаграмма класса User

1.2.2.1 Конструктор

1) User()

1.2.2.2 Методы класса

- 1) add_user()
- 2) remove_user()
- 3) find_user()
- 4) clear_users()

1.2.2.3 Методы доступа к компонентам класса

- 1) set_user_id()
- 2) set_username()
- 3) set_user_icon()
- 4) get_user_id()
- 5) get_user_name()

1.2.3 Класс Server_Settings



Рис. 5: UML-диаграмма класса MainWindow

1.2.3.1 Конструктор

- 1) MainWindow()

1.2.3.2 Методы класса

- 1) sendMessage()
- 2) createMenu()
- 3) userListUpdated()
- 4) saveXMLFile()
- 5) log()
- 6) makeElement()

1.3 Приложение Client

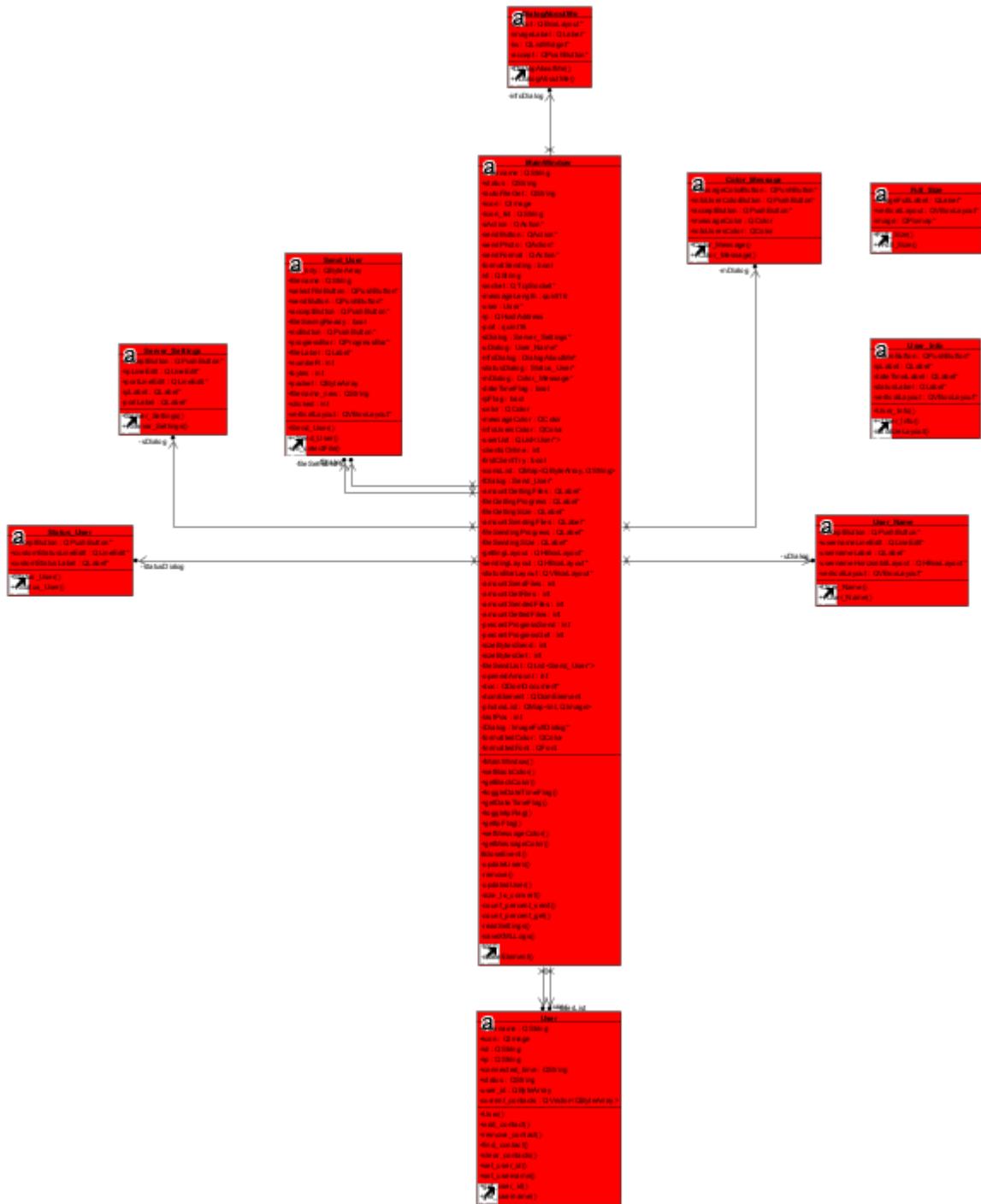


Рис. 6: UML-диаграмма приложения Client

1.3.1 Класс DialogAboutMe

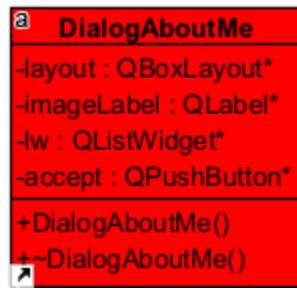


Рис. 7: UML-диаграмма класса DialogAboutMe

1.3.1.1 Конструктор

- 1) DialogAboutMe()

1.3.1.2 Деструктор

- 1) ~DialogAboutMe()

1.3.2 Класс Full_Size

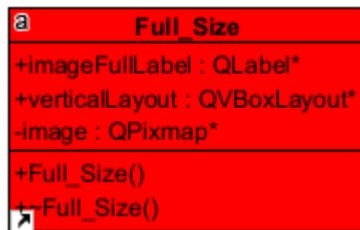


Рис. 8: UML-диаграмма класса Full_Size

1.3.2.1 Конструктор

- 1) Full_Size()

1.3.2.2 Деструктор

- 1) ~Full_Size()

1.3.3 Класс User_Name

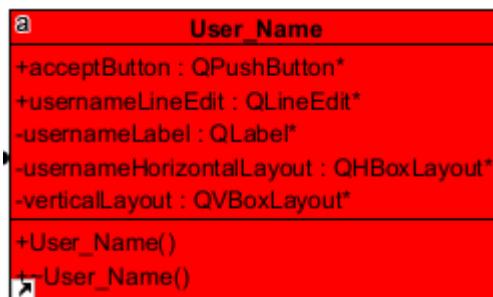


Рис. 9: UML-диаграмма класса User_Name

1.3.3.1 Конструктор

1) User_Name()

1.3.3.2 Деструктор

1) ~User_Name()

1.3.4 Класс Color_Message

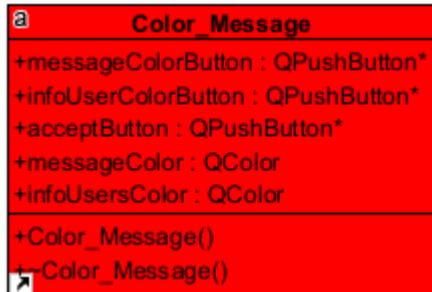


Рис. 10: UML-диаграмма класса Color_Message

1.3.4.1 Конструктор

1) Color_Message()

1.3.4.2 Деструктор

1) ~Color_Message()

1.3.5 Класс Send_User

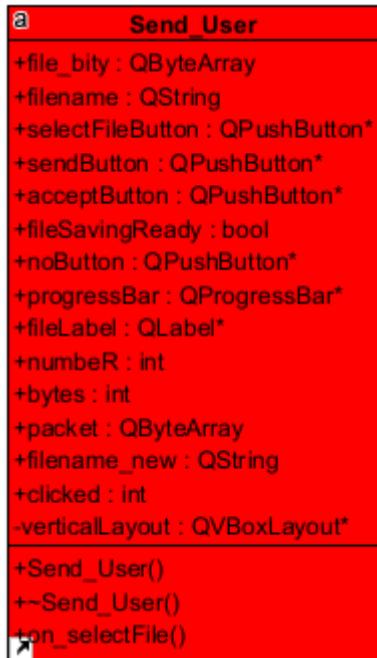


Рис. 11: UML-диаграмма класса Send_User

1.3.5.1 Конструктор

1) Send_User()

1.3.5.2 Деструктор

1) ~Send_User()

1.3.5.3 Методы класса

1) on_selectFiler()

1.3.6 Класс Server_Settings



Рис. 12: UML-диаграмма класса Server_Settings

1.3.6.1 Конструктор

1) Server_Settings()

1.3.6.2 Деструктор

1) ~Server_Settings()

1.3.7 Класс Status_User

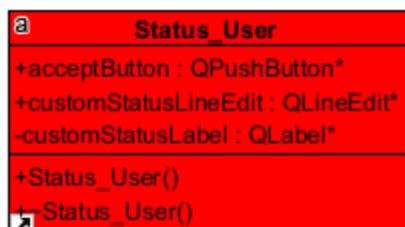


Рис. 13: UML-диаграмма класса Status_User

1.3.7.1 Конструктор

1) Status_User()

1.3.7.2 Деструктор

1) ~Status_User()

1.3.8 Класс User_Info

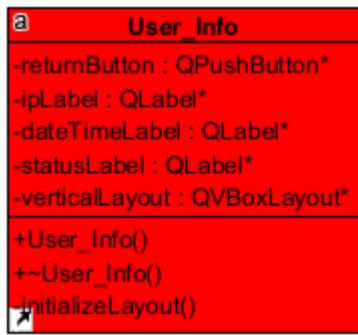


Рис. 14: UML-диаграмма класса User_Info

1.3.8.1 Конструктор

- 1) User_Info()

1.3.8.2 Деструктор

- 1) ~User_Info()

1.3.8.3 Методы класса

- 1) Initialize_Layout()

1.3.9 Класс User

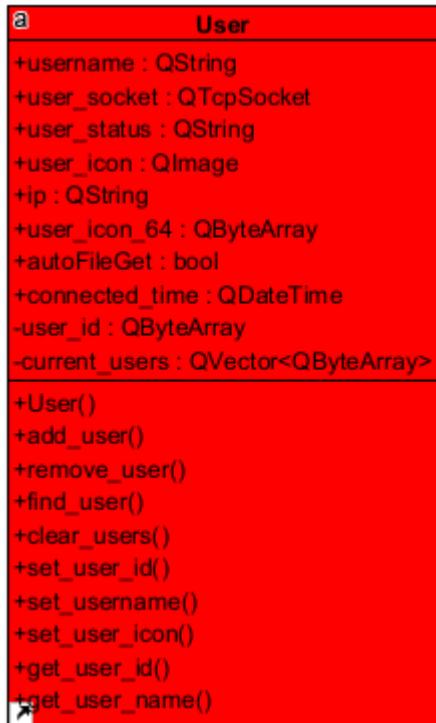


Рис. 15: UML-диаграмма класса User

1.3.9.1 Конструктор

- 1) User()

1.3.9.2 Методы класса

- 1) add_user()
- 2) remove_user()
- 3) find_user()
- 4) clear_users()

1.3.9.3 Методы доступа к компонентам класса

- 1) set_user_id()
- 2) set_username()
- 3) set_user_icon()
- 4) get_user_id()
- 5) get_user_id()

1.3.10 Класс MainWindow

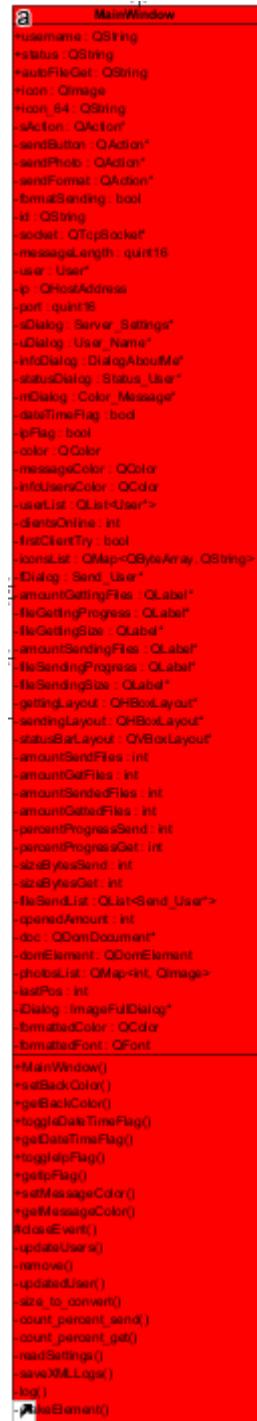


Рис. 16: UML-диаграмма класса MainWindow

1.3.10.1 Конструктор

- 1) MainWindow()

1.3.10.2 Методы класса

- 1) toggleDateTimeFlag()
- 2) toggleHelpFlag()
- 3) closeEvent()
- 4) updateUsers()
- 5) remove()
- 6) size_to_convert()
- 7) count_percent_send()
- 8) count_percent_get()
- 9) readSettings()
- 10) saveXMLLogs()
- 11) log()
- 12) makeElement()

1.3.10.3 Методы класса

- 1) setBackColor()
- 2) getBackColor()
- 3) getDateFlag()
- 4) getIpFlag()

2. Выполнение задач

2.1 Приложение XML Reader

2.1.1 Конструктор

MainWindow() – конструктор, создающий окно приложения, в котором настраивается стиль оформления окна, а также устанавливает ui форму.

2.1.2 Методы слоты класса

on_action_Exit_triggered() – закрытый метод слот класса, открывающий диалоговое окно выбора XML файла и помещающий содержимое этого файла во внутреннюю область центрального виджета.

on_action_Exit_2_triggered() – закрытый метод слот класса, который выполняет выход из данного приложения.

2.1.3 Деструктор

~MainWindow() – деструктор , выполняющий удаление ui формы.

2.2 Приложение Server

2.2.1 Класс Server_Settings

2.2.1.1 Конструктор

Server_Settings() – конструктор, генерирующий новое окно с областями ввода для двух полей класса – ip и port. Выполняется также проверка, чтобы поля были непустыми.

2.2.1.2 Деструктор

~Server_Settings() – деструктор по умолчанию, который реализуется поведением компилятора (=default). В случае не ввода данных, поля будут заполнены автоматически компилятором.

2.2.2 Класс User

2.2.2.1 Конструктор

User() – конструктор по умолчанию, который реализуется поведением компилятора (=default). В случае не ввода данных, поля будут заполнены автоматически компилятором.

2.2.2.2 Методы класса

add_user() – метод, выполняющий добавление данных в вектор, которые содержат информацию о пользователе.

remove_user() – метод, выполняющий удаление данных из вектора, который содержит информацию о пользователях.

find_user() – метод, выполняющий поиск данных из вектора, который содержит информацию о пользователях.

clear_users() – метод, выполняющий очистку вектора, который содержит информацию о пользователях.

2.2.2.3 Методы получения компонентов класса

set_user_id() – метод, выполняющий задание параметров пользователю.

set_username() – метод, выполняющий задание имени пользователю.

set_user_icon() – метод, выполняющий задание иконки пользователю.

get_user_id() – метод, возвращающий параметры пользователя.

get_user_name() – метод, возвращающий имя пользователя.

2.2.3 Класс MainWindow

2.2.3.1 Конструктор

MainWindow() – конструктор, выполняющий создание всего окна, вызовом другого метода `createMenu()`, а также устанавливает тёмную тему приложения.

2.2.3.2 Методы класса

sendMessage() – открытый метод, выполняющий оповещение об подключении/отключении пользователей.

createMenu() – закрытый метод, создание меню главного окна.

userListUpdated() – закрытый метод, выполняющий обновление текущих данных о пользователях на сервере.

saveXMLFile() – закрытый метод, выполняющий сохранение логов в отдельный xml файл.

log() – закрытый метод, выполняющий создание логов на сервере.

makeElement() – закрытый метод, выполняющий отображение логов на главном экране.

2.3 Приложение Client

2.3.1 Класс DialogAboutMe

2.3.1.1 Конструктор

DialogAboutMe() – конструктор данного класса, выполняющий заполнение диалогового окна сведениями об авторе и Qt.

2.3.1.2 Деструктор

~DialogAboutMe() – деструктор по умолчанию.

2.3.2 Класс Full_Size

2.3.2.1 Конструктор

Full_Size() – конструктор данного класса, выполняющий создание окна, с помещением в это окно исходного изображения.

2.3.2.2 Деструктор

~Full_Size() – деструктор по умолчанию.

2.3.3 Класс User_Name

2.3.3.1 Конструктор

User_Name() – конструктор данного класса, создающий окно с областью ввода данных об имени пользователя.

2.3.3.2 Деструктор

~User_Name() – деструктор по умолчанию.

2.3.4 Класс Color_Message

2.3.4.1 Конструктор

Color_Message() – конструктор данного класса, создающий окно с выбором цвета отображаемых сообщений на клиенте.

2.3.4.2 Деструктор

~Color_Message() - деструктор по умолчанию.

2.3.5 Класс Send_User

2.3.5.1 Конструктор

Send_User() – конструктор данного класса, создающий окно с кнопками выбора файла и отправки файла, а также шкалу загрузки файла.

2.3.5.2 Деструктор

~Send_User() - деструктор по умолчанию.

2.3.5.3 Методы класса

on_SelectFile() – метод открывающий диалоговое окно проводника, предлагающее пользователю выбрать текстовый файл для отправки.

2.3.6 Класс Server_Settings

2.3.6.1 Конструктор

Server_Settings() – конструктор, генерирующий новое окно с областями ввода для двух полей класса – ip и port. Выполняется также проверка, чтобы поля были непустыми.

2.3.6.2 Деструктор

~Server_Settings() – деструктор по умолчанию, который реализуется поведением компилятора (=default). В случае не ввода данных, поля будут заполнены автоматически компилятором.

2.3.7 Класс Status_User

2.3.7.1 Конструктор

Status_User() – конструктор, создающий окно с областью ввода своего пользовательского статуса.

2.3.7.2 Деструктор

~Status_User() – деструктор по умолчанию, который реализуется поведением компилятора (=default). В случае не ввода данных, поля будут заполнены автоматически компилятором.

2.3.8 Класс User_Info

2.3.8.1 Конструктор

User_Info() – конструктор, создающий окно с информацией о пользователе.

2.3.8.2 Деструктор

~User_Info() – деструктор по умолчанию, который реализуется поведением компилятора (=default). В случае не ввода данных, поля будут заполнены автоматически компилятором.

2.3.8.3 Методы класса

InitializeLayout() – метод, выполняющий компоновку окна с информацией о пользователе.

2.3.8 Класс User

2.3.8.1 Конструктор

User() – конструктор по умолчанию, который реализуется поведением компилятора (=default). В случае не ввода данных, поля будут заполнены автоматически компилятором.

2.3.8.2 Методы класса

add_user() – метод, выполняющий добавление данных в вектор, которые содержат информацию о пользователе.

remove_user() – метод, выполняющий удаление данных из вектора, который содержит информацию о пользователях.

find_user() – метод, выполняющий поиск данных из вектора, который содержит информацию о пользователях.

clear_users() – метод, выполняющий очистку вектора, который содержит информацию о пользователях.

2.3.8.3 Методы получения компонентов класса

set_user_id() – метод, выполняющий задание параметров пользователю.

set_username() – метод, выполняющий задание имени пользователю.

set_user_icon() – метод, выполняющий задание иконки пользователю.

get_user_id() – метод, возвращающий параметры пользователя.

get_user_name() – метод, возвращающий имя пользователя.

2.3.9 Класс MainWindow

2.3.9.1 Конструктор

MainWindow() – конструктор, создающий окно клиента, инициализируя ui файл.

2.3.9.2 Методы класса

closeEvent() – перегрузка виртуальной функции, не позволяющее самопроизвольно закрывать приложение без сохранения настроек приложения.

updateUsers() – закрытый метод класса, обновляющий информацию о подключенных к серверу клиентов.

remove() – закрытый метод класса, выполняющий очистку главного экрана от дочерних окон.

updatedUser () – закрытый метод класса, обновляющий информацию о подключенном к серверу клиенту.

size_to_convert () – закрытый метод класса, обновляющий информацию в статус баре о размере файла.

count_percent_send() – закрытый метод класса, обновляющий информацию о передаваемом файле в процентах.

count_percent_get() – закрытый метод класса, обновляющий информацию о получаемом файле в процентах.

readSettings() – закрытый метод класса, устанавливающий настройки приложения.

saveXMLLogs() – закрытый метод класса, сохраняющий историю переписки в отдельный xml файл.

log() – закрытый метод класса, создающий логи клиента внутри приложения.

makeElement() – закрытый метод класса, отображающий логи клиента в окне слева.

2.3.9.3 Методы получения компонентов класса

setBackColor() – открытый метод класса, задающий цвет окна слева.

getBackColor() – открытый метод класса, возвращающий цвет окна слева.

toggleDateTimeFlag() – открытый метод класса, переключающий отображение времени сообщения.

getDateTimeFlag() – открытый метод класса, возвращающий состояние булевского значения, отвечающий за отображение времени сообщения.

toggleIpFlag() – открытый метод класса, переключающий отображение IP в сообщениях.

getIpFlag() – открытый метод класса, возвращающий состояние булевского значения, отвечающий за отображение IP в сообщениях.

setMessageColor() – открытый метод класса, задающий цвет отображаемых сообщений.

getMessageColor() – открытый метод класса, возвращающий цвет отображаемых сообщений.

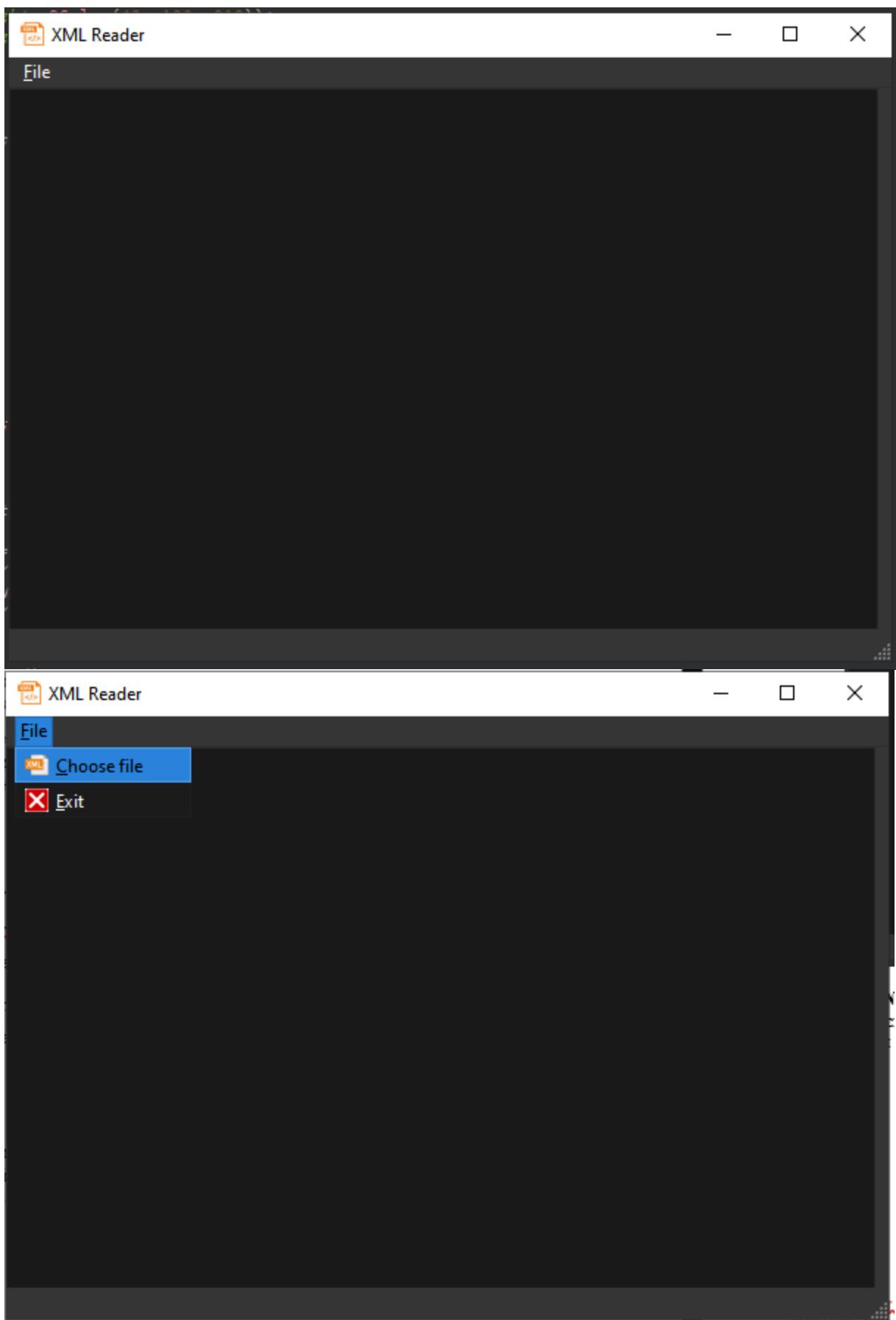
3. Получение исполняемых модулей

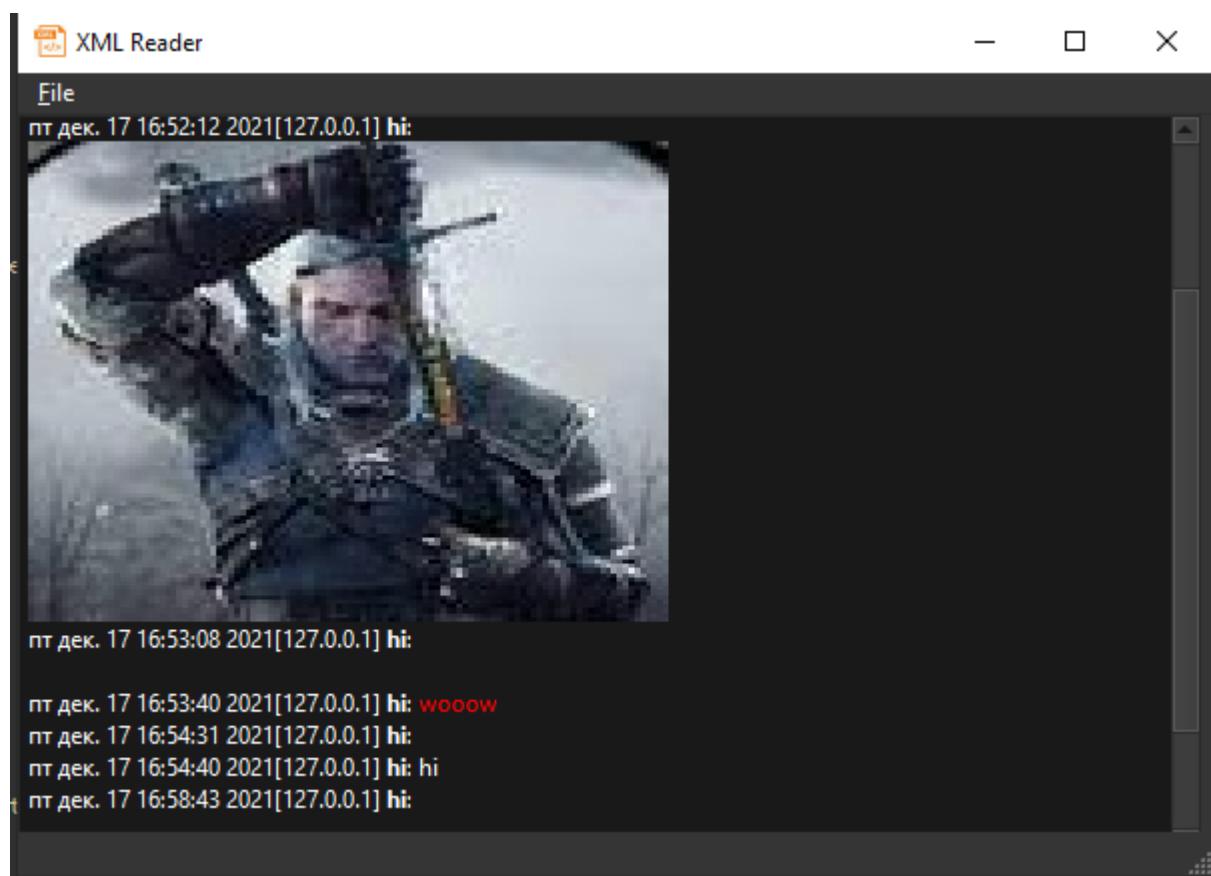
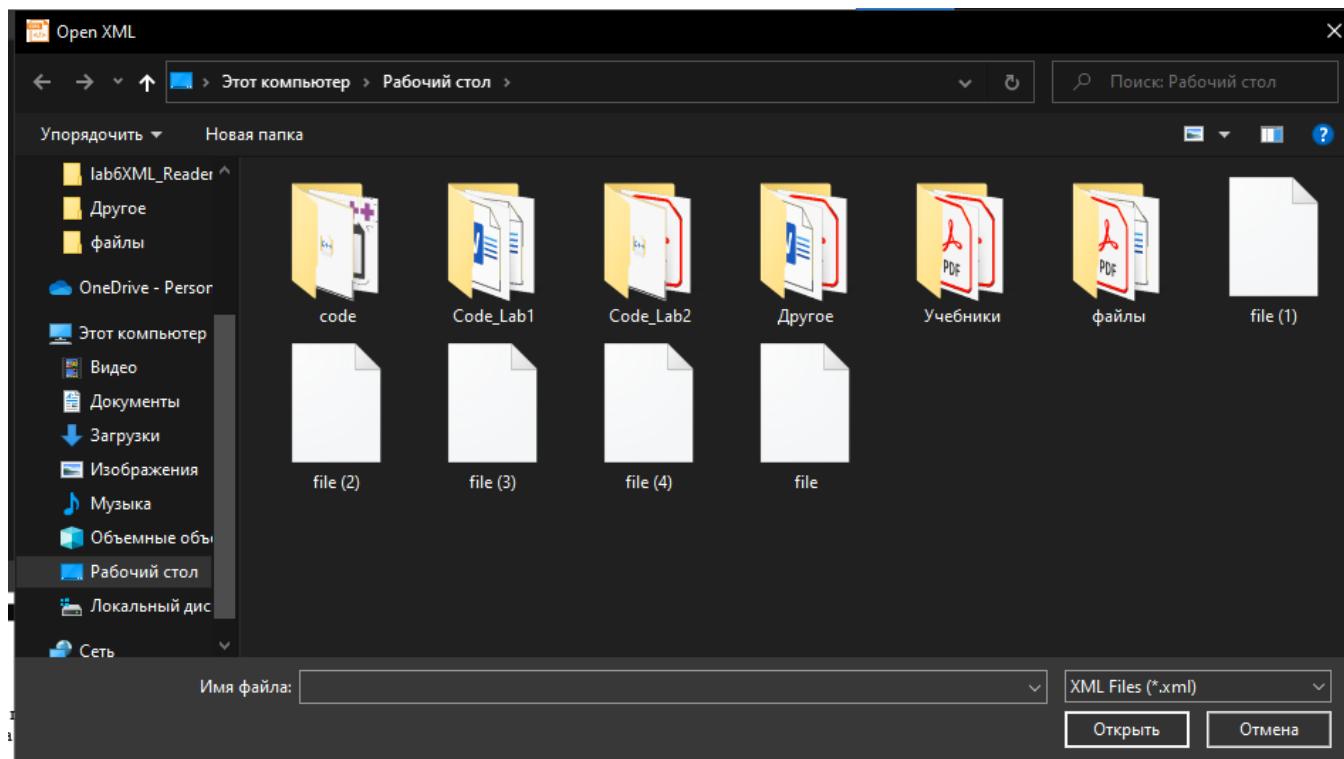
С помощью системы сборки qmake производится компоновка (сборка) исполняемого модуля из объектных модулей. Объектные модули – файлы, которые уже могут быть запущены, получаются, соответственно, из файлов исходного кода.

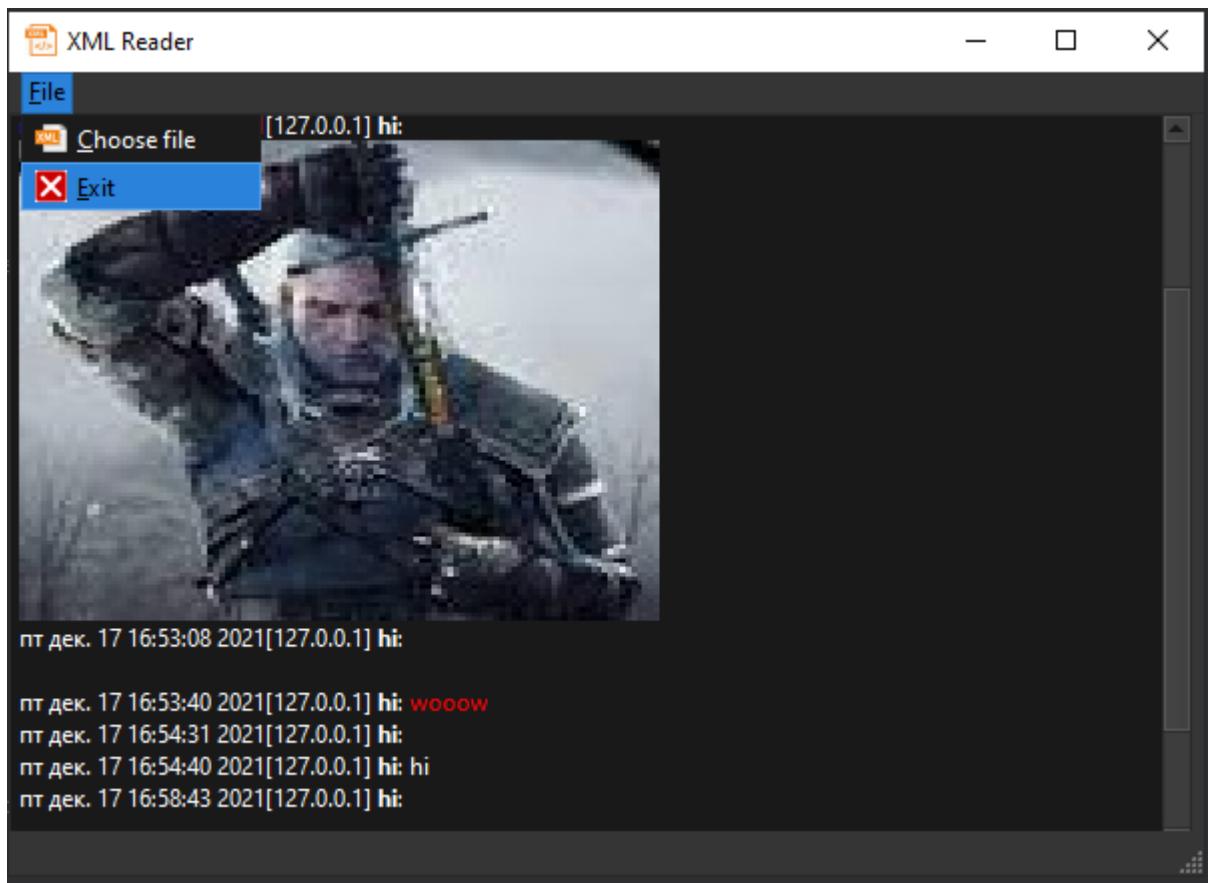
4. Тестирование

1) Test №1

Происходит проверка всех возможных функций приложения.

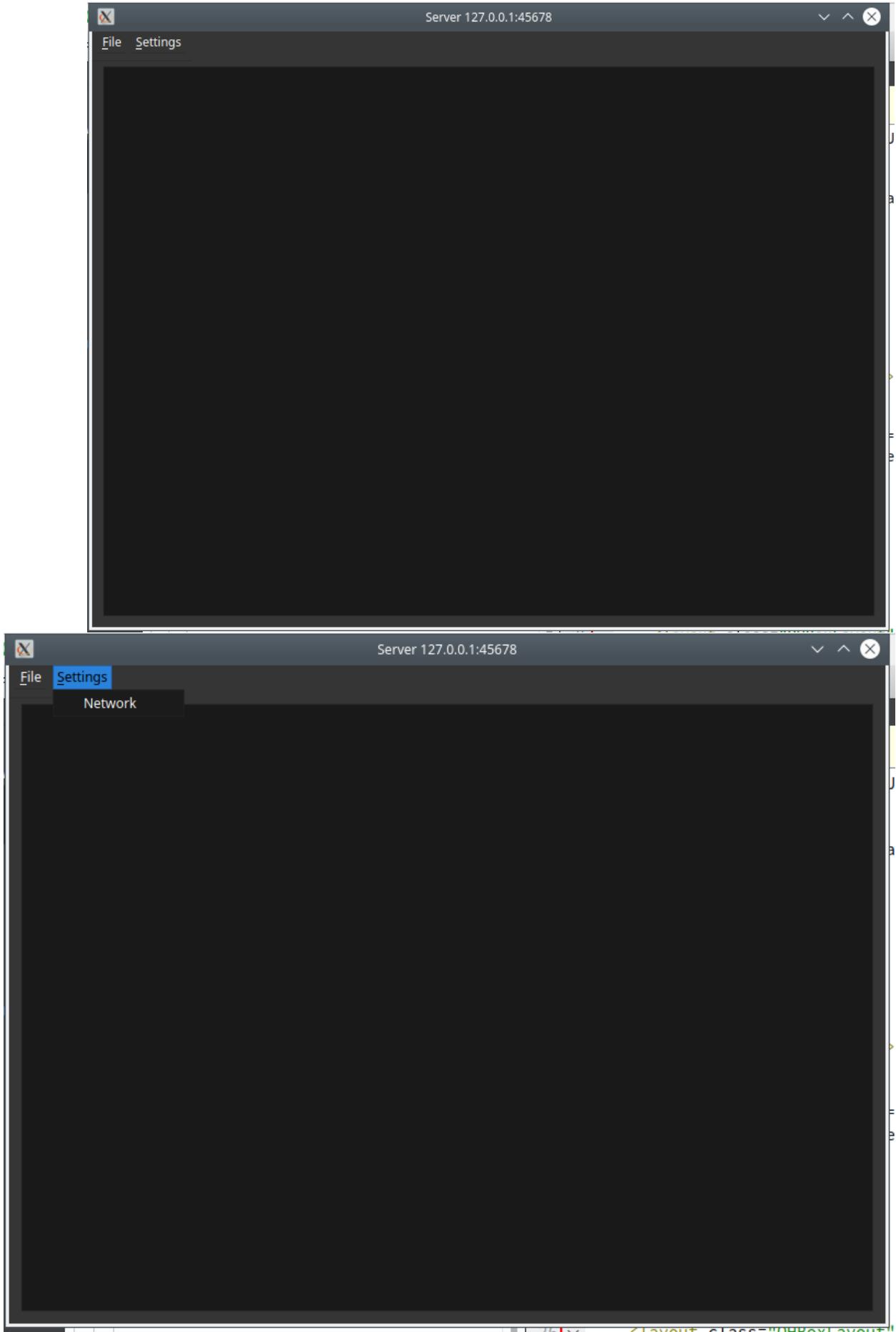


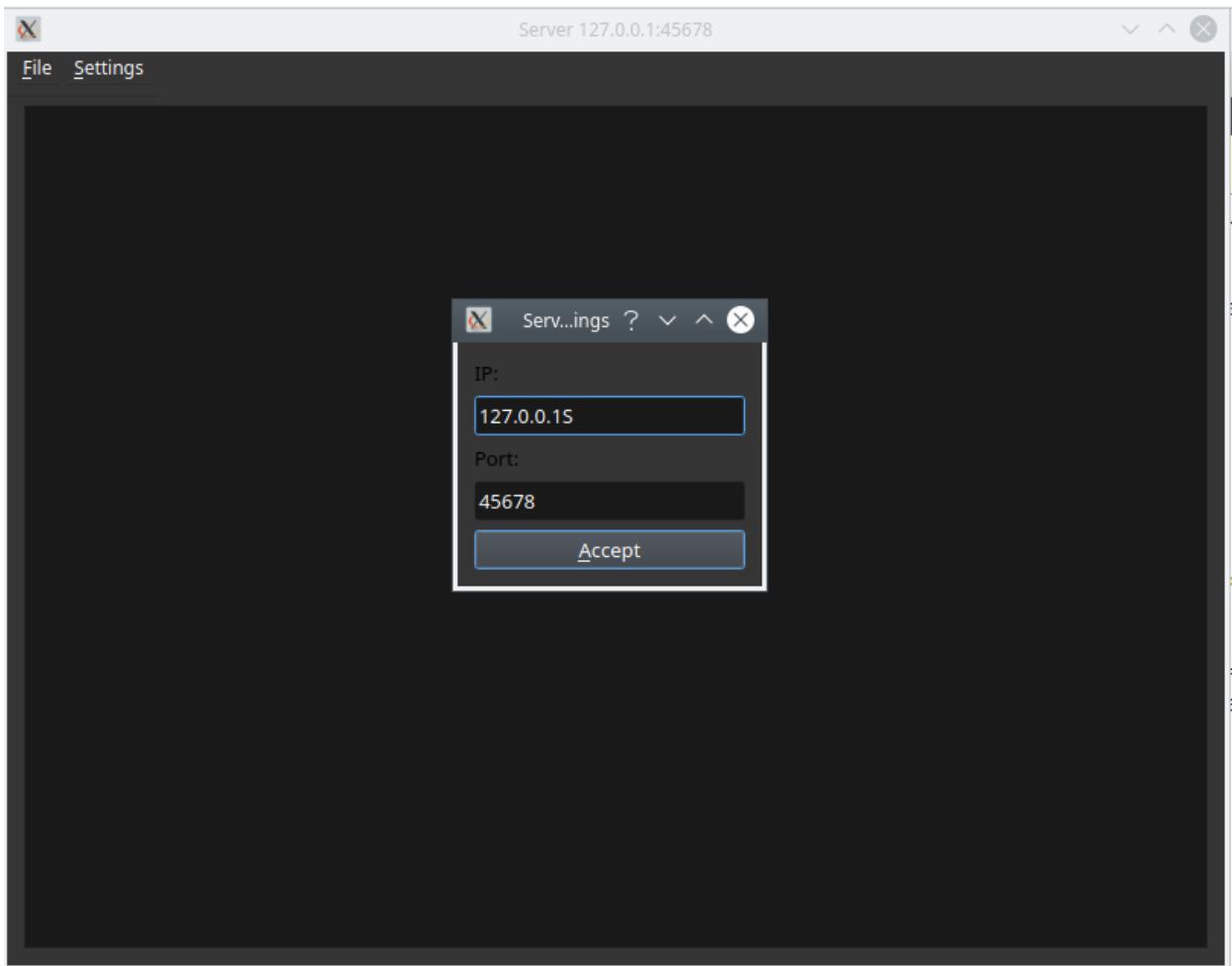


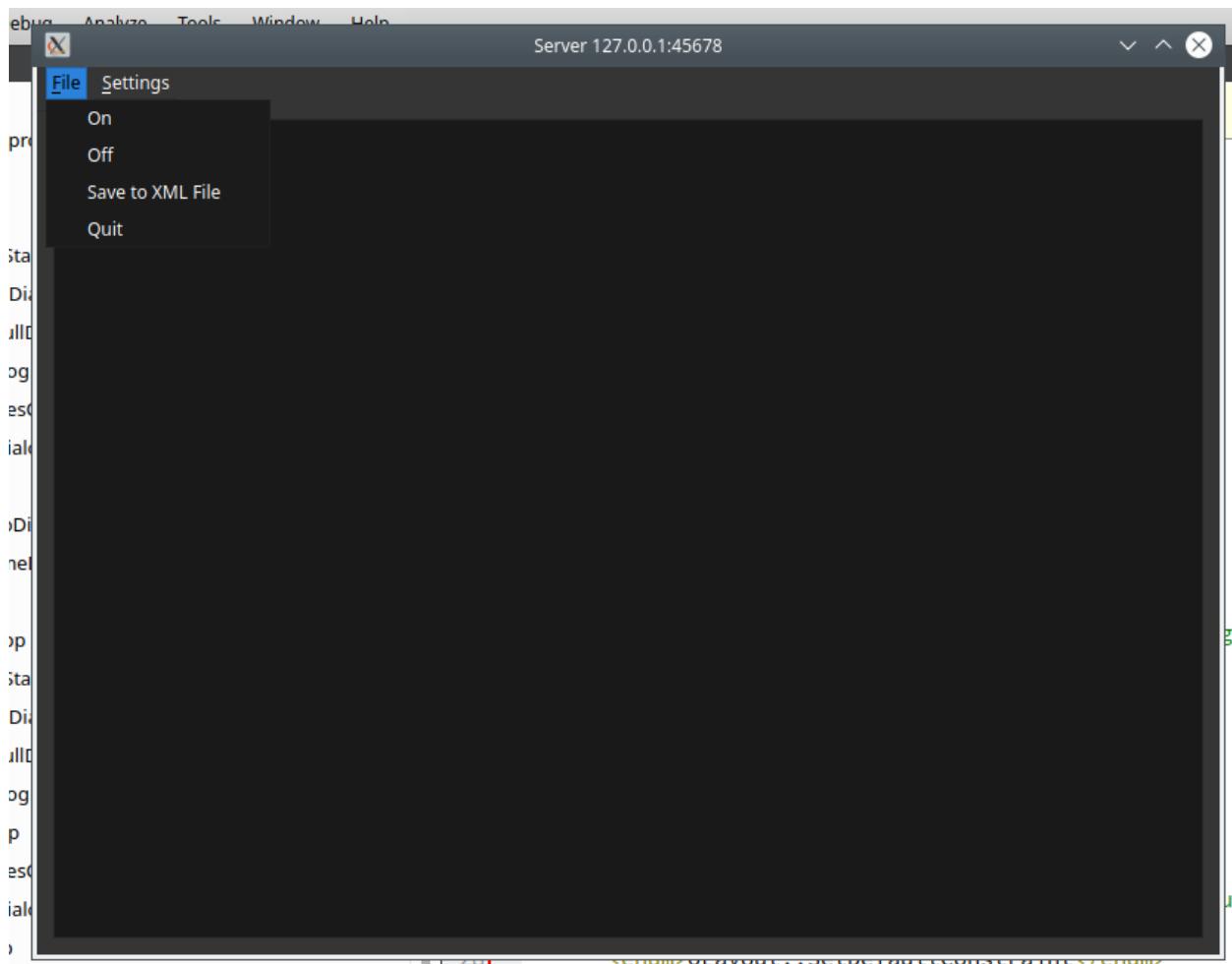


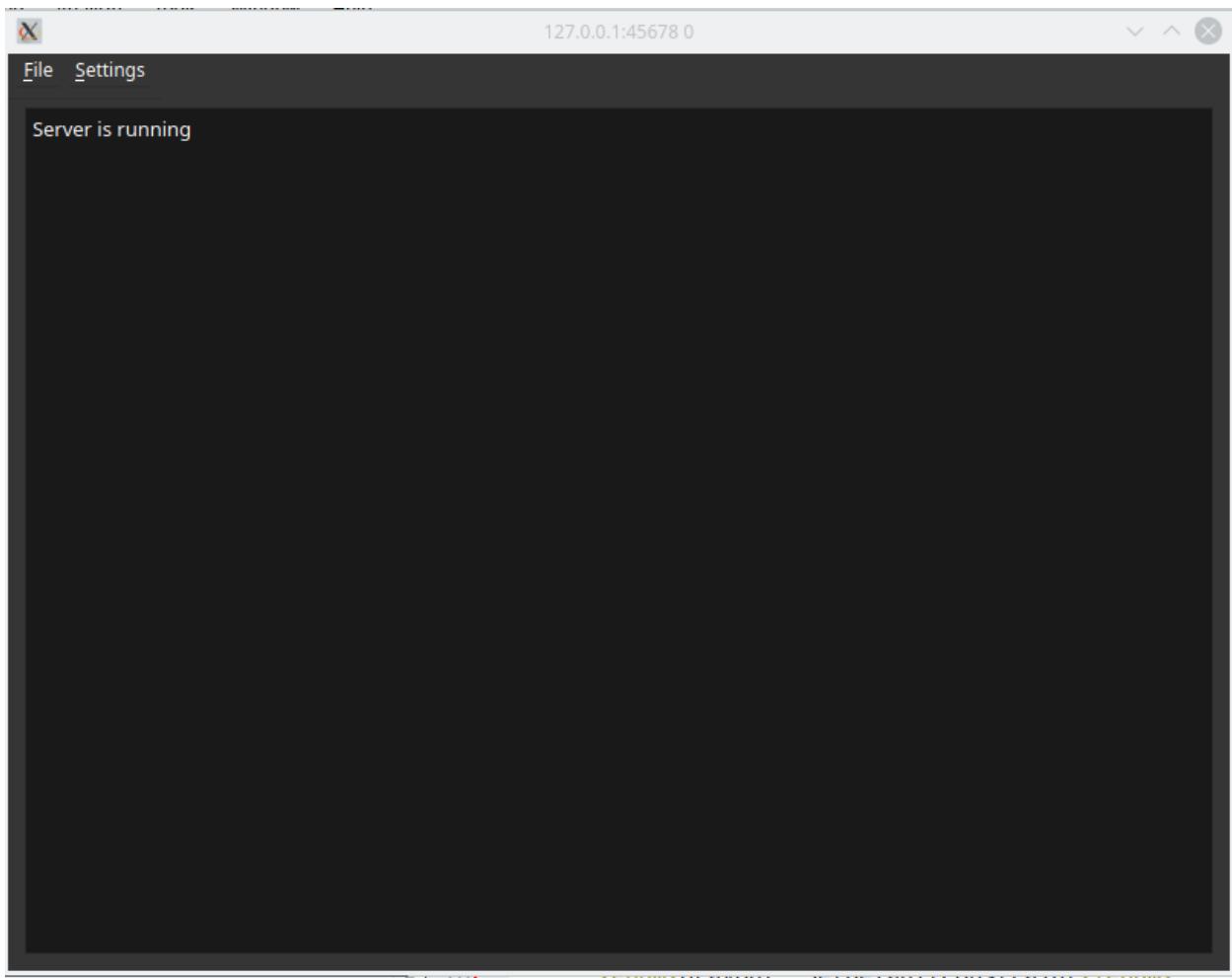
2) Test №2

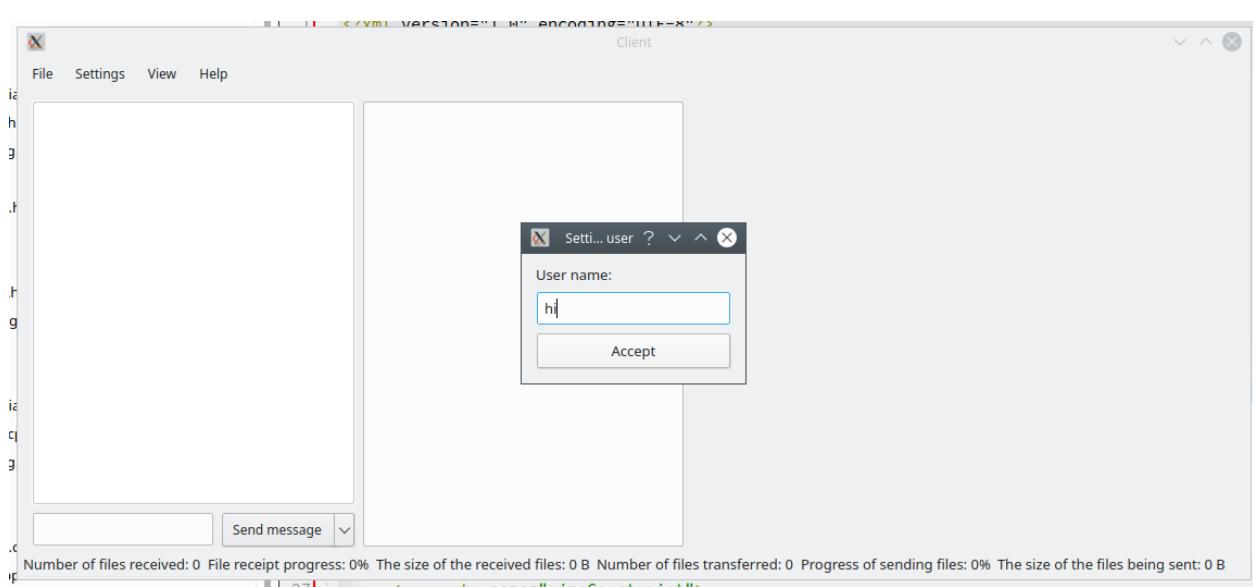
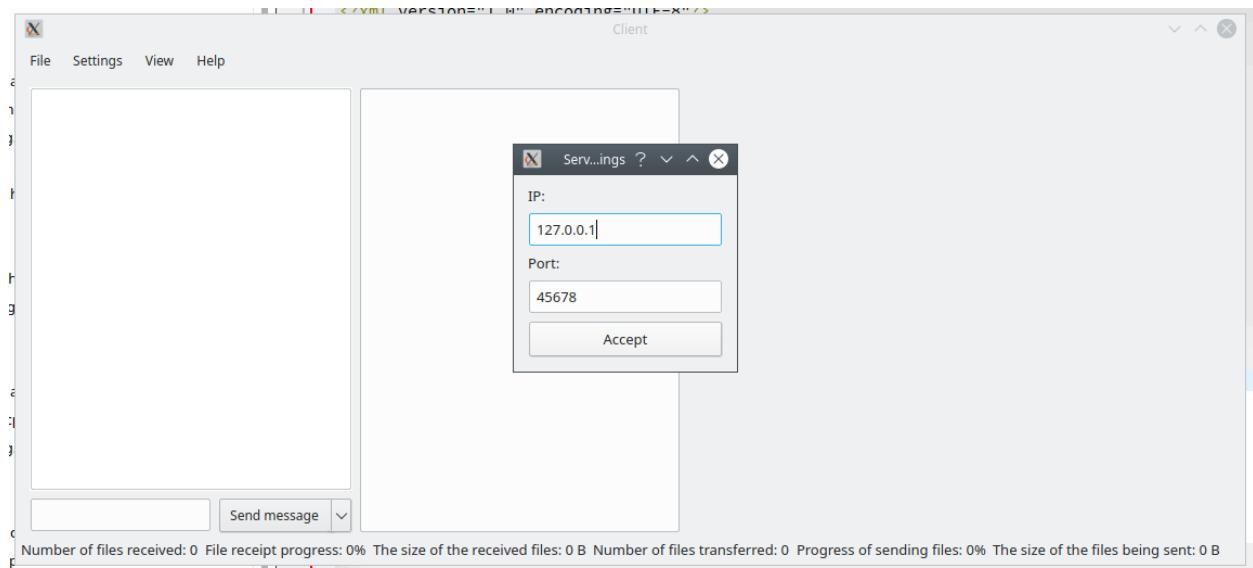
Происходит совместное тестирование клиента и сервера. Также идёт проверка основного функционала.

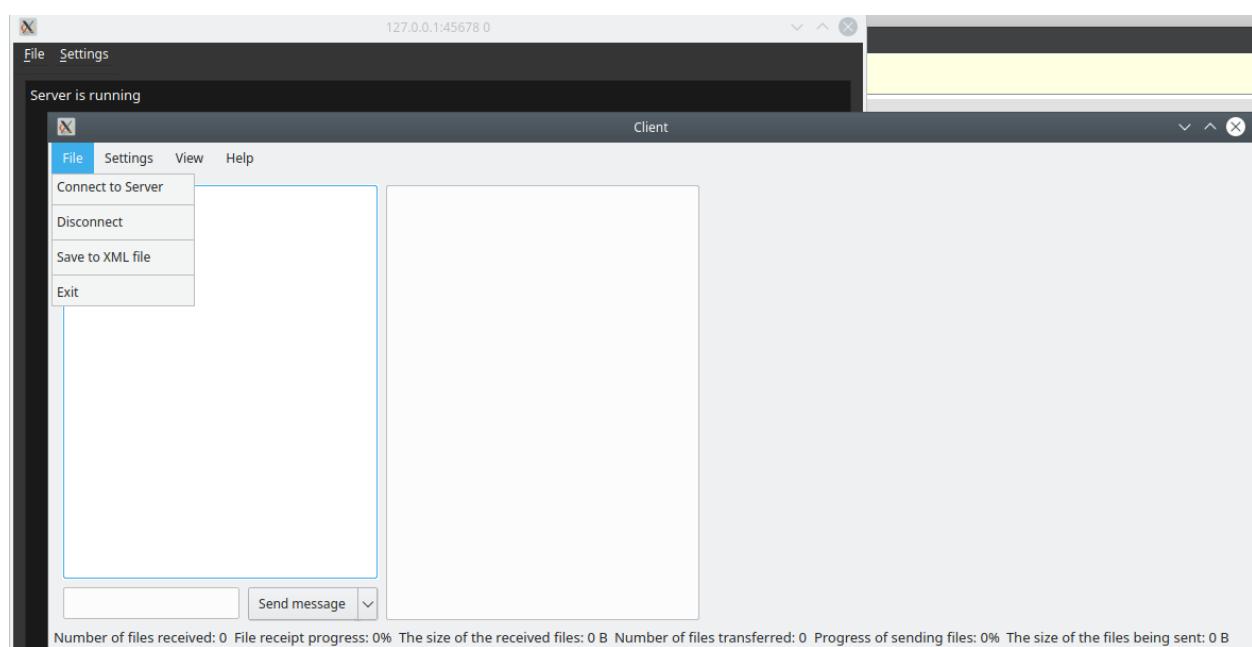
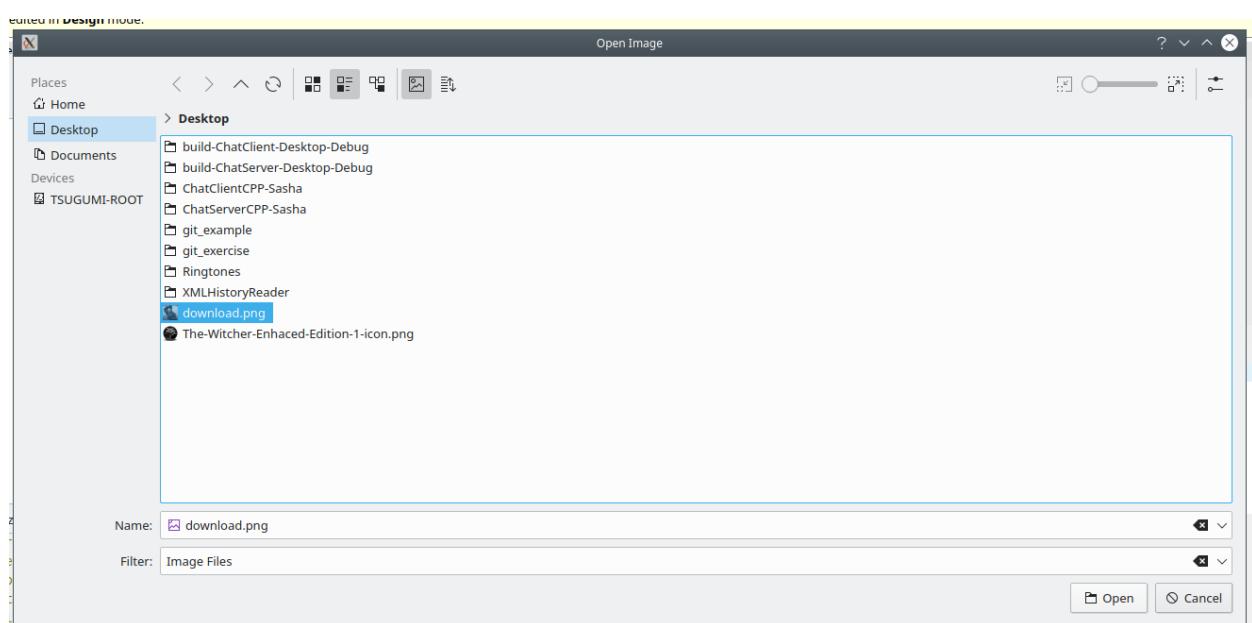
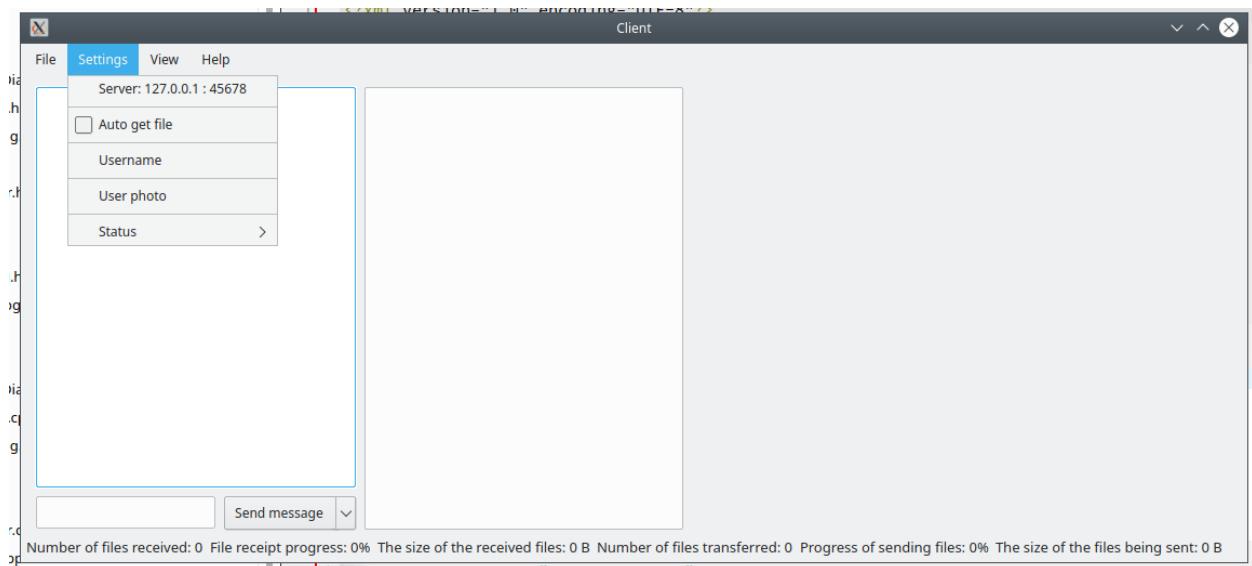


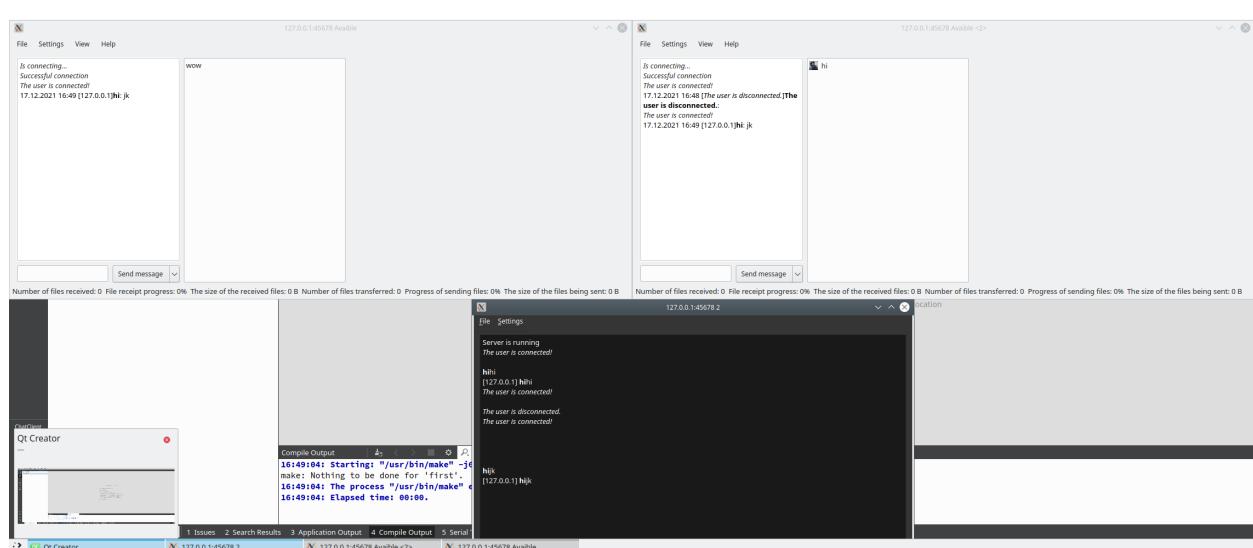
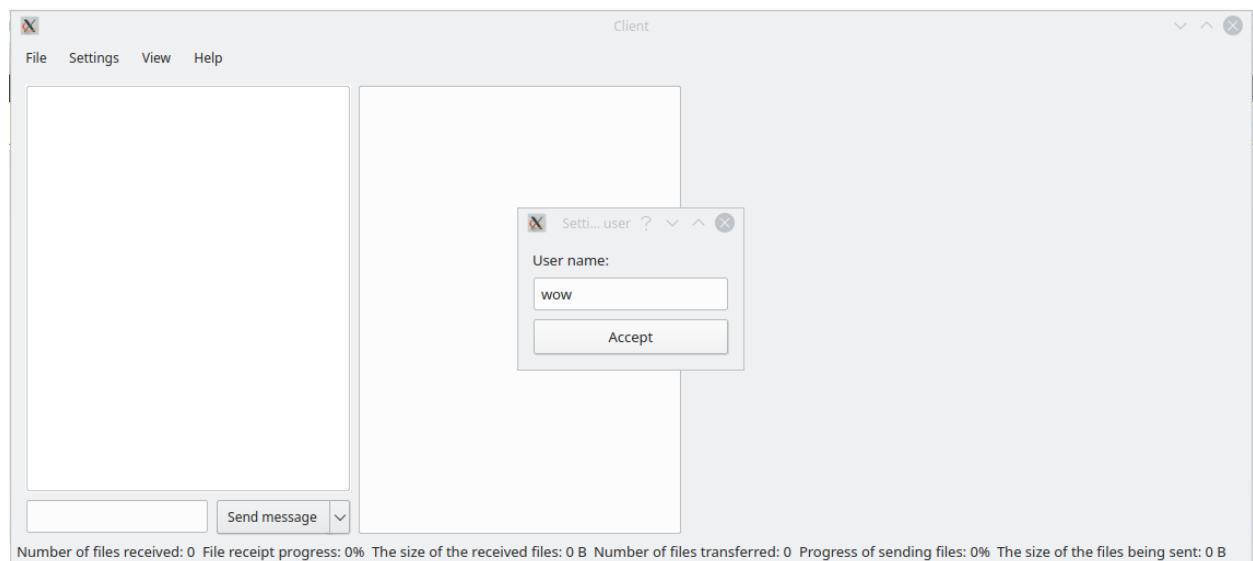
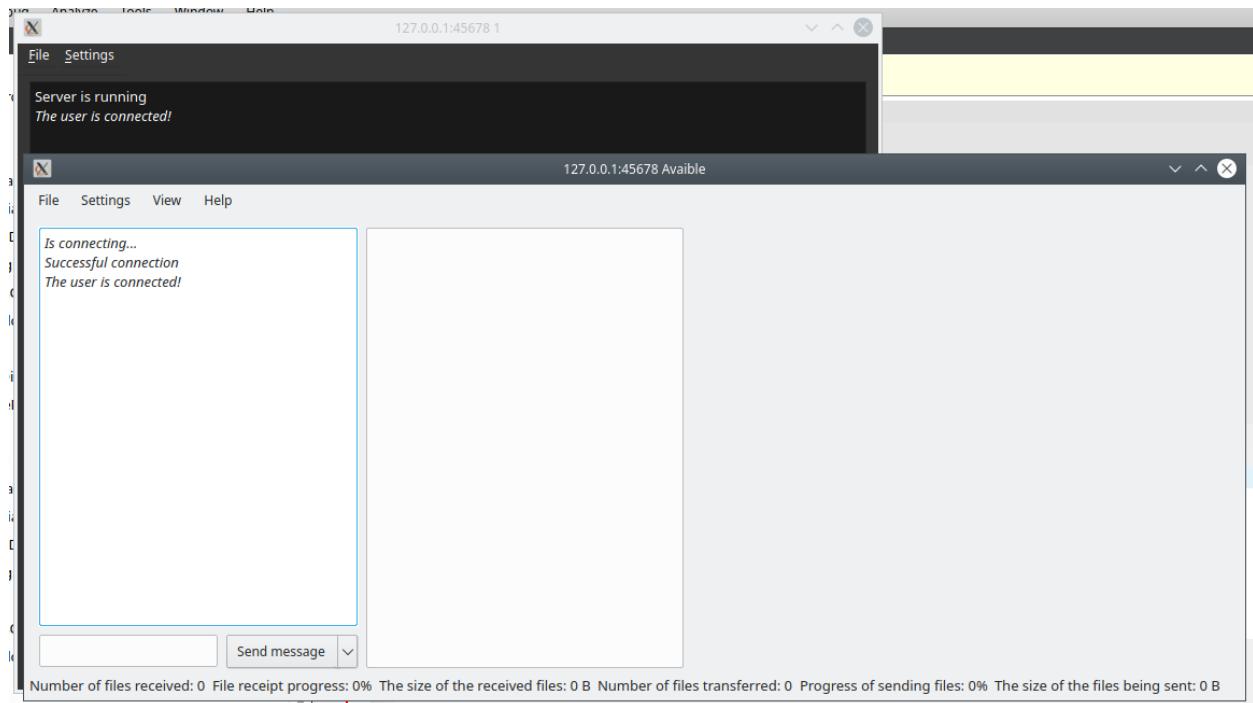


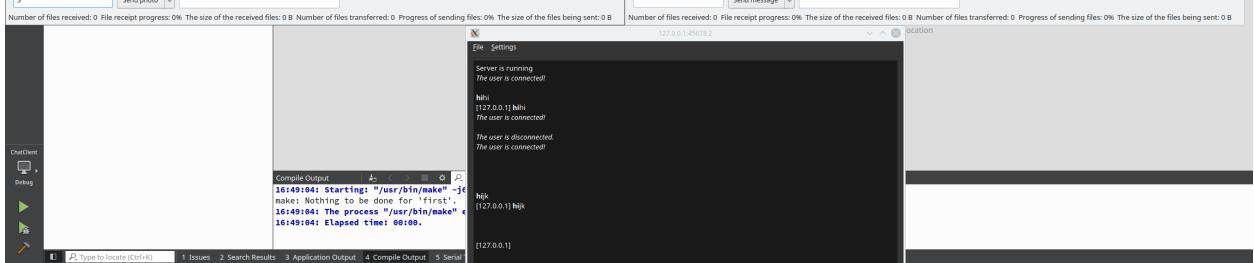
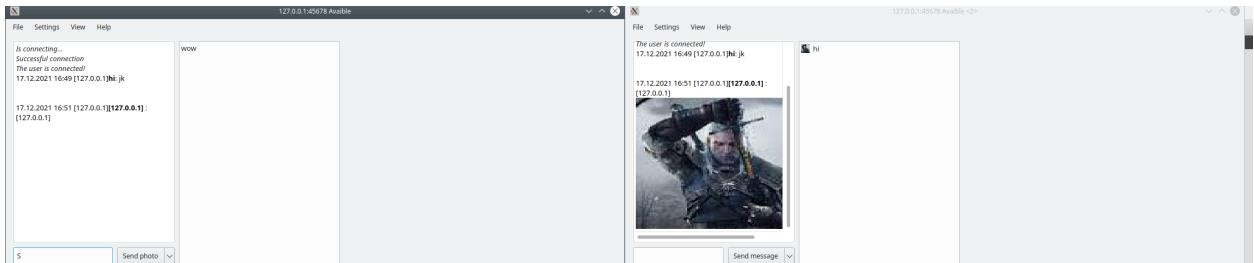
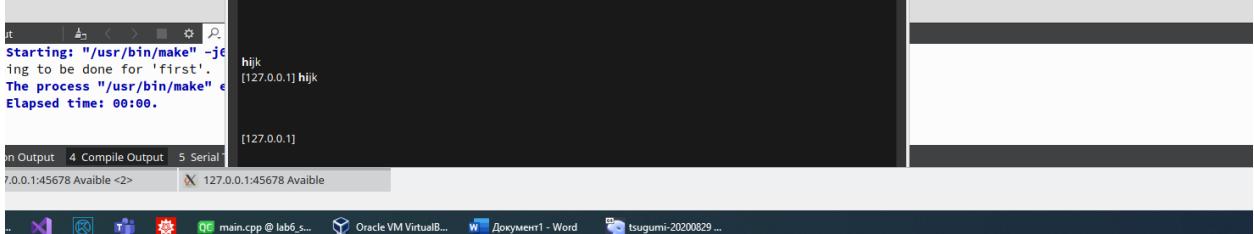
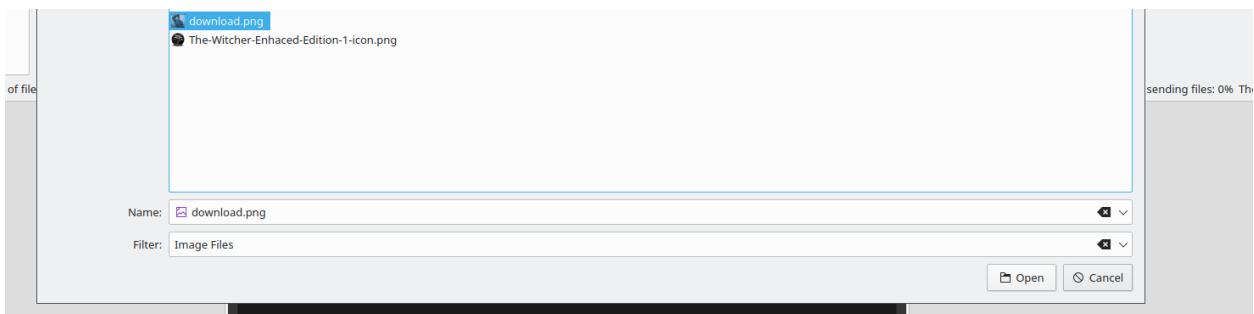
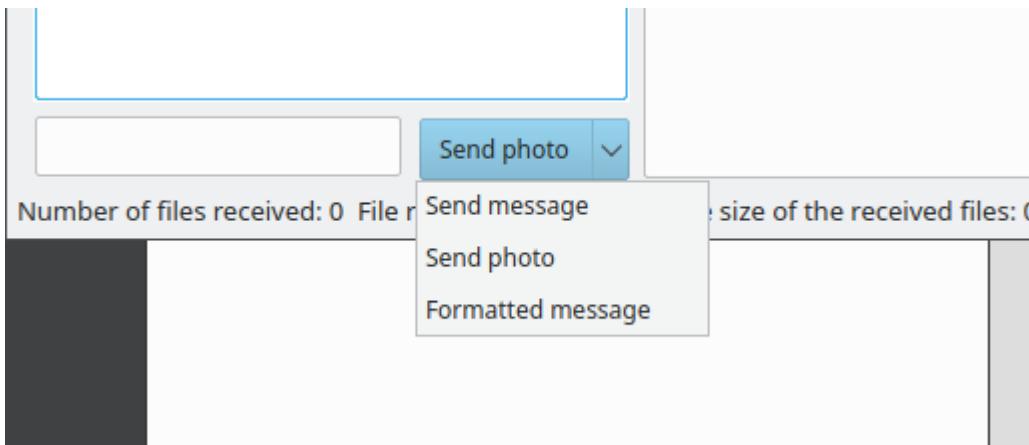


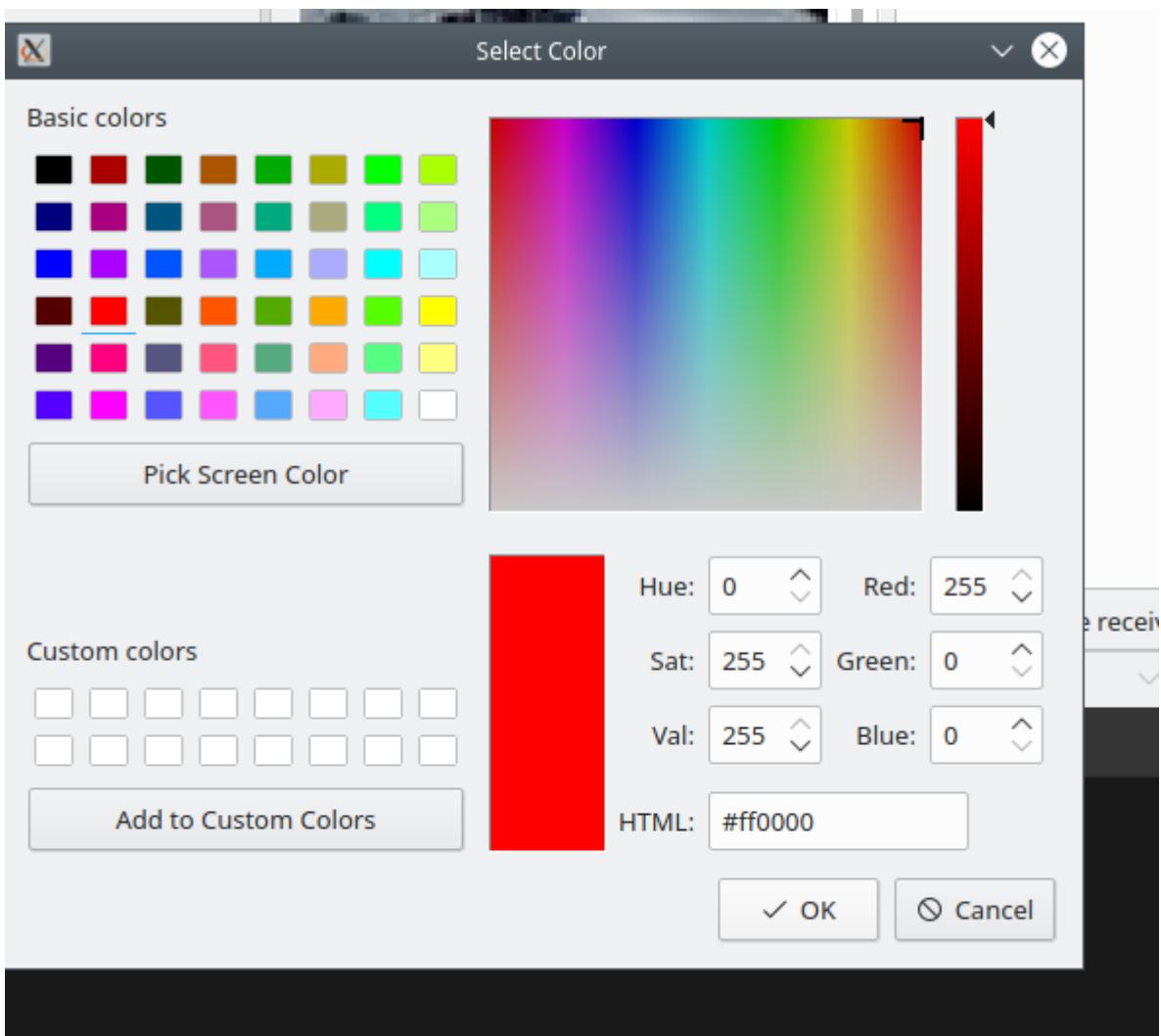
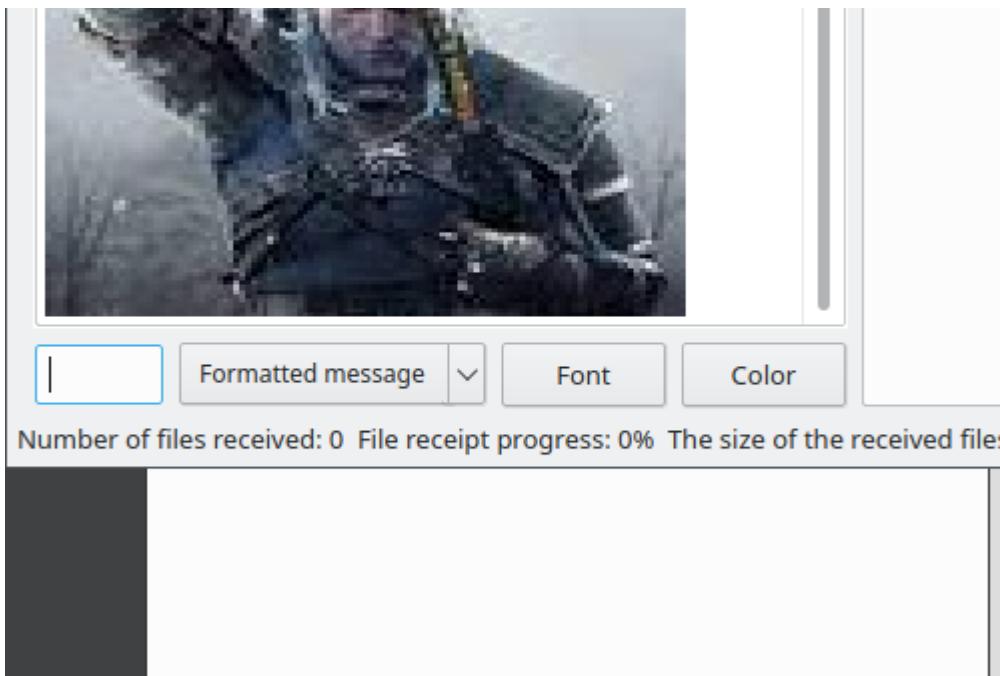


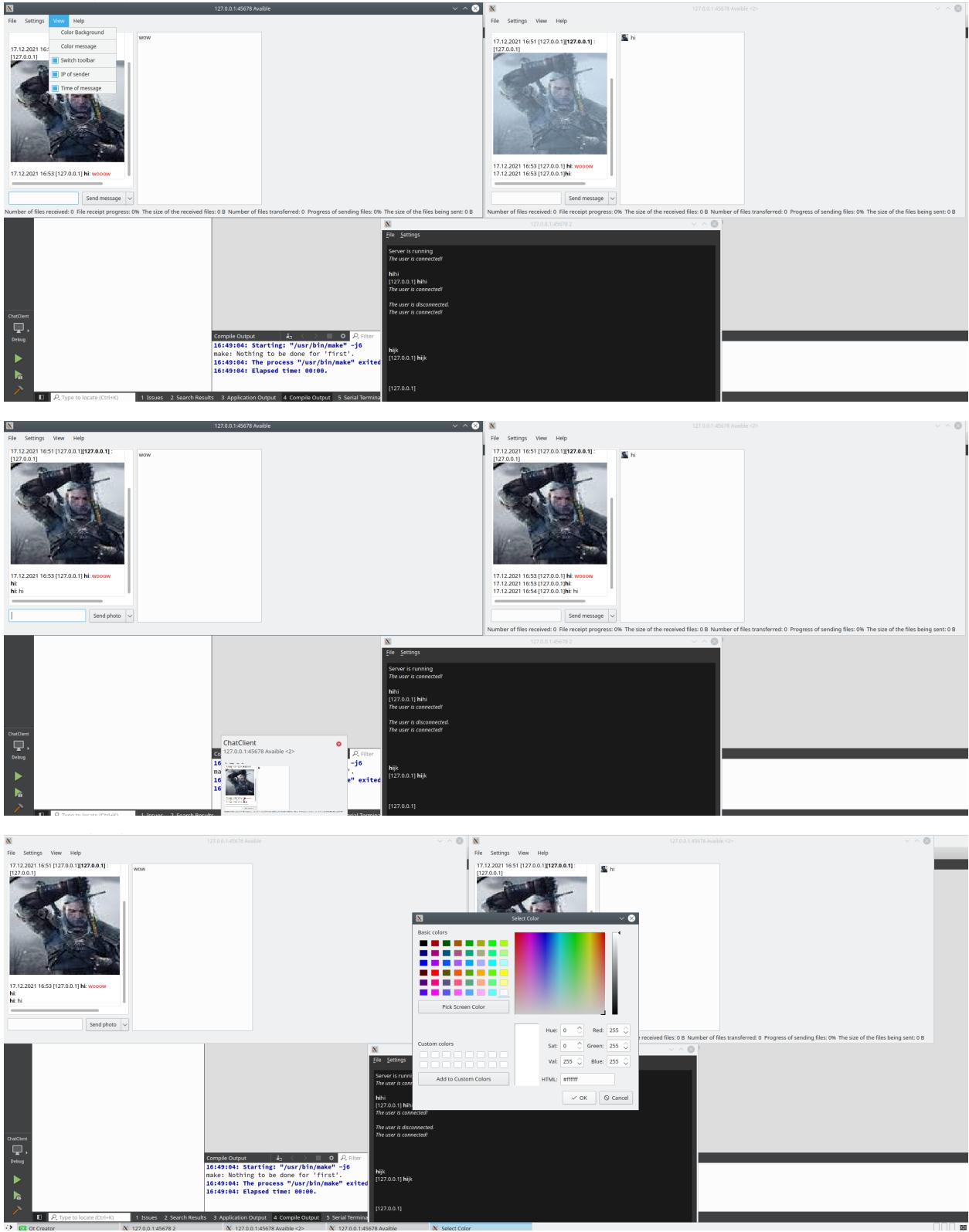


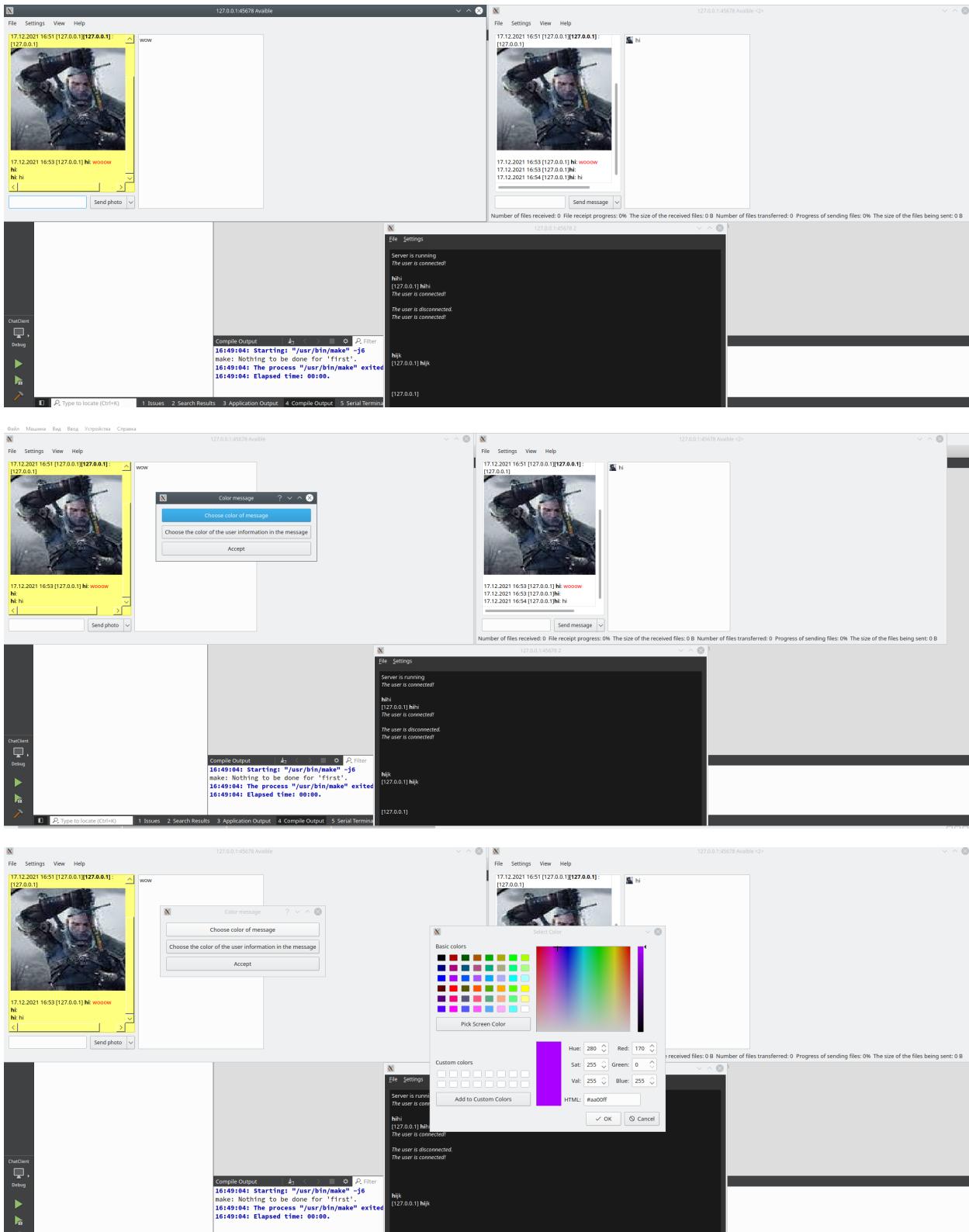


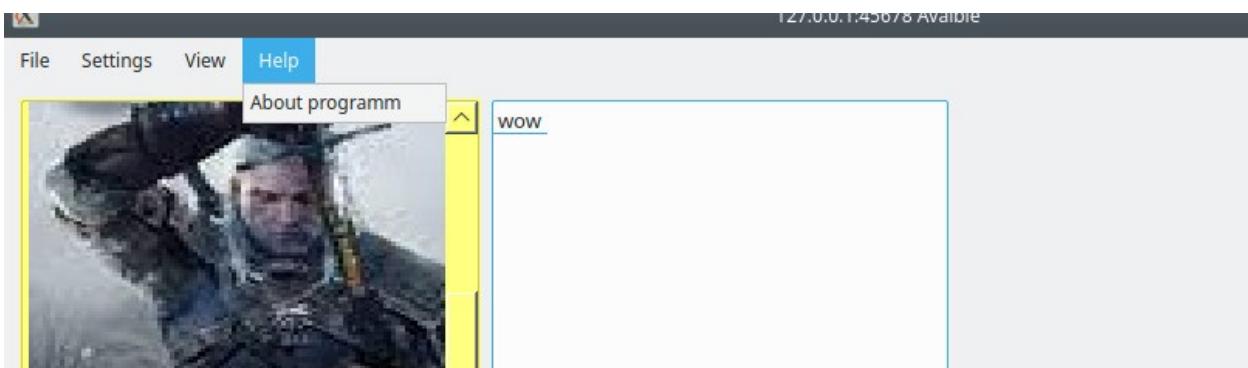
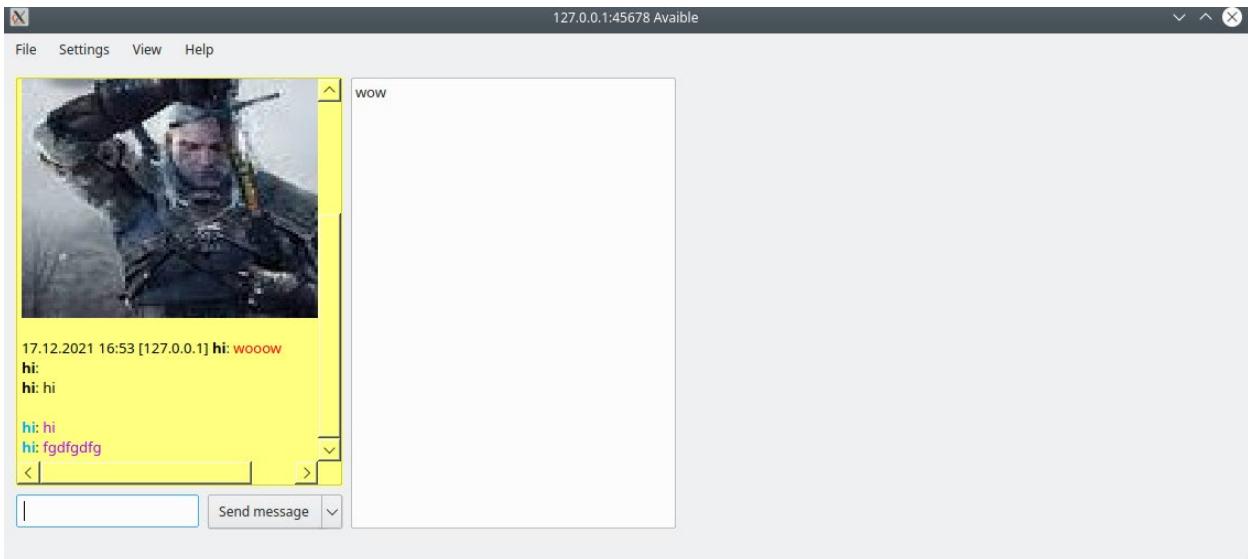
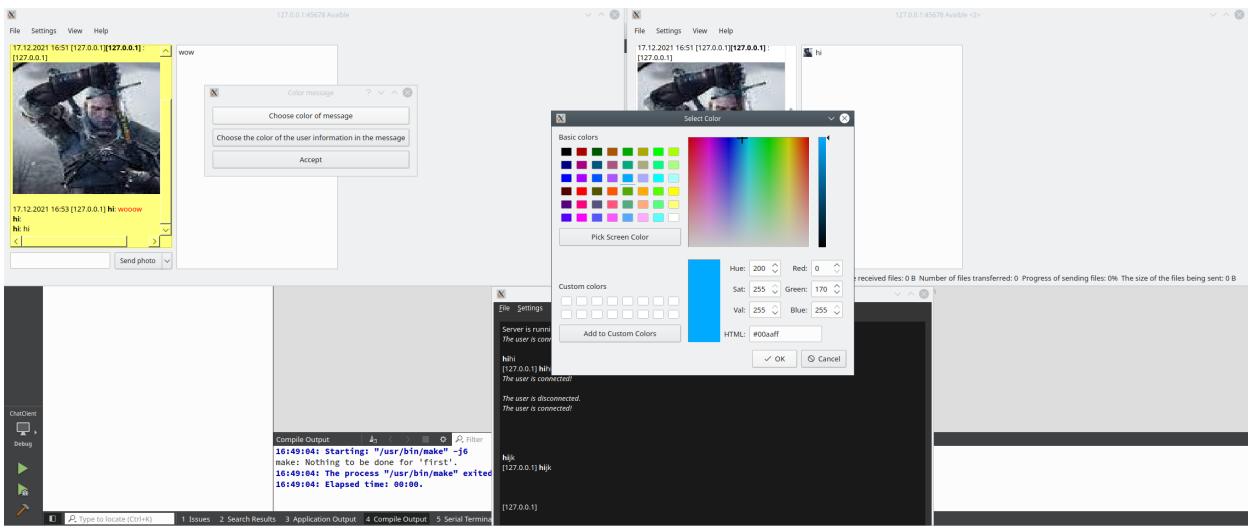


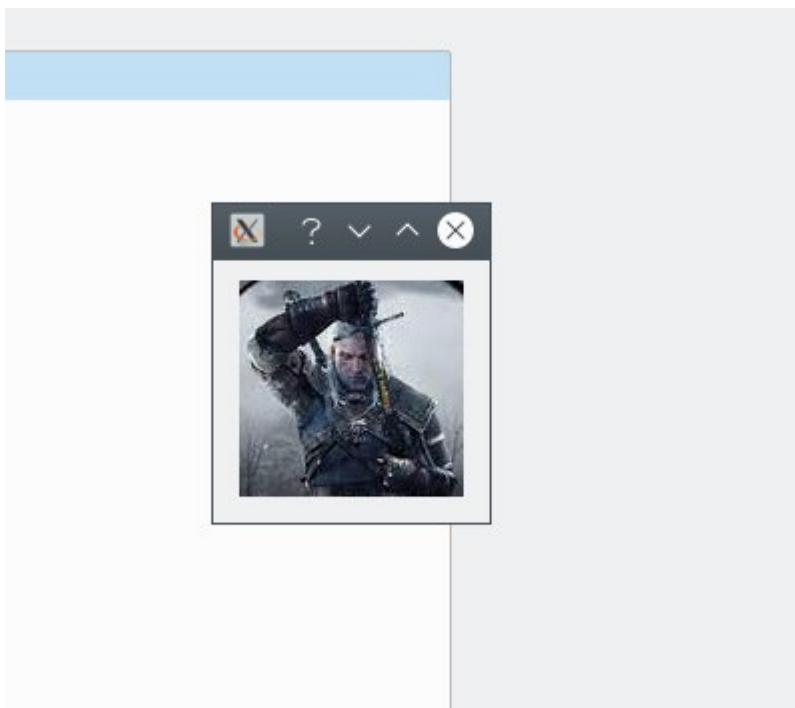
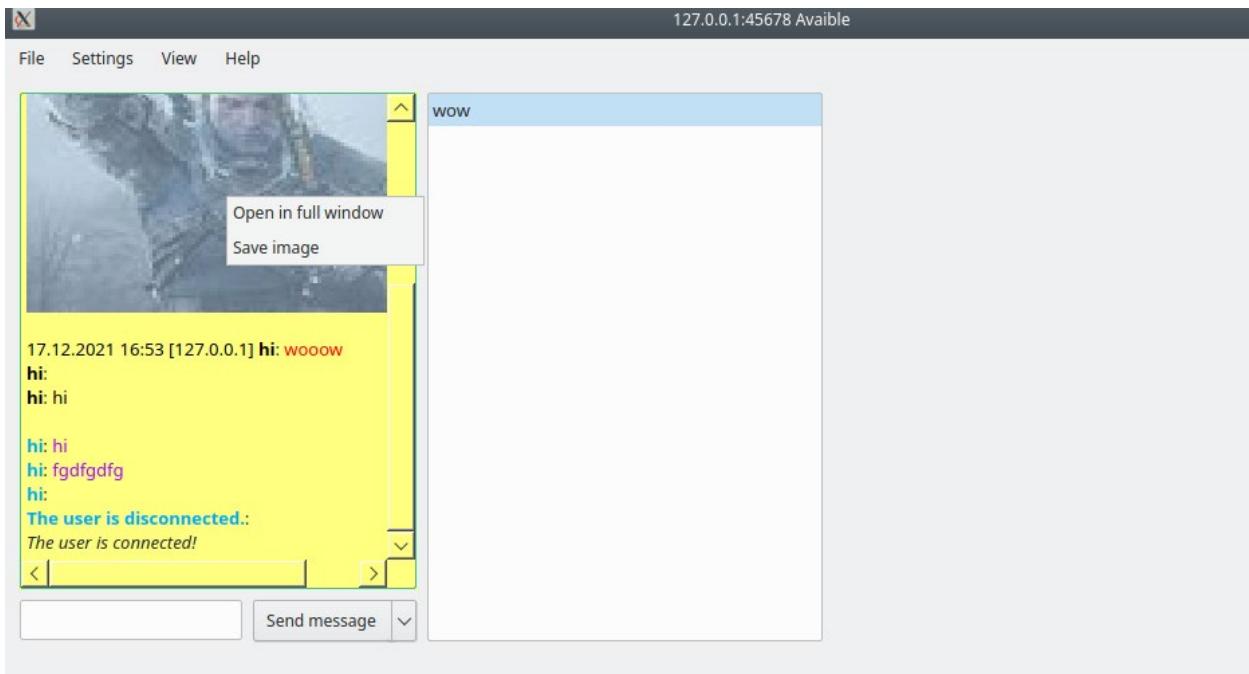


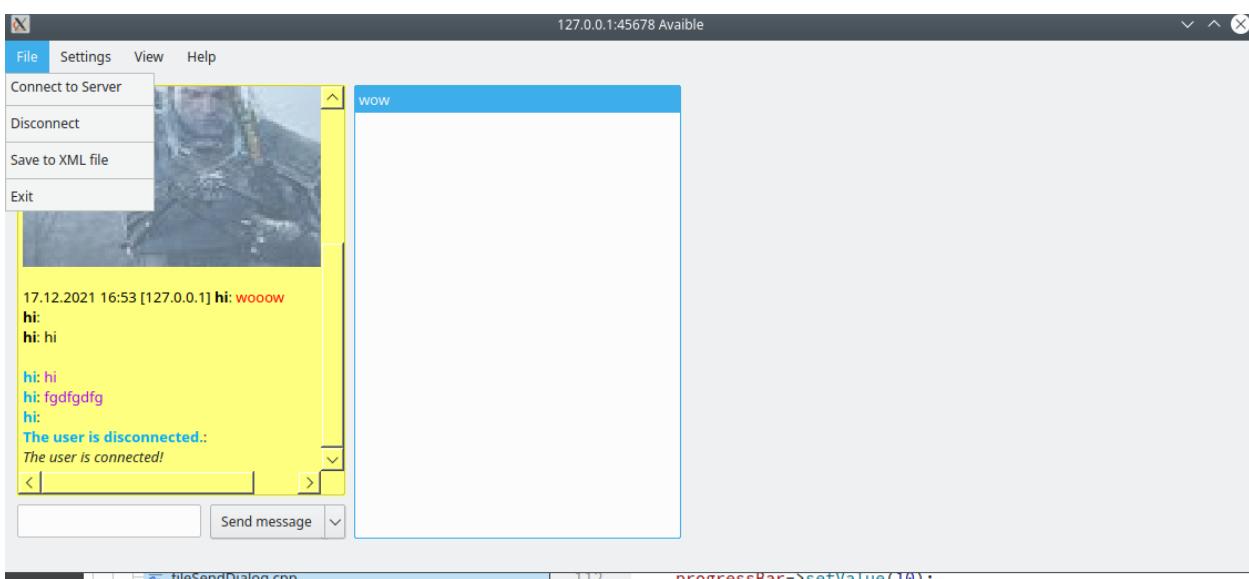
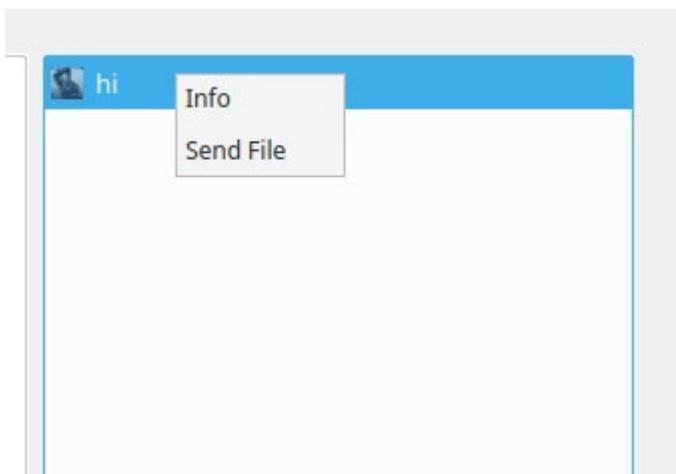
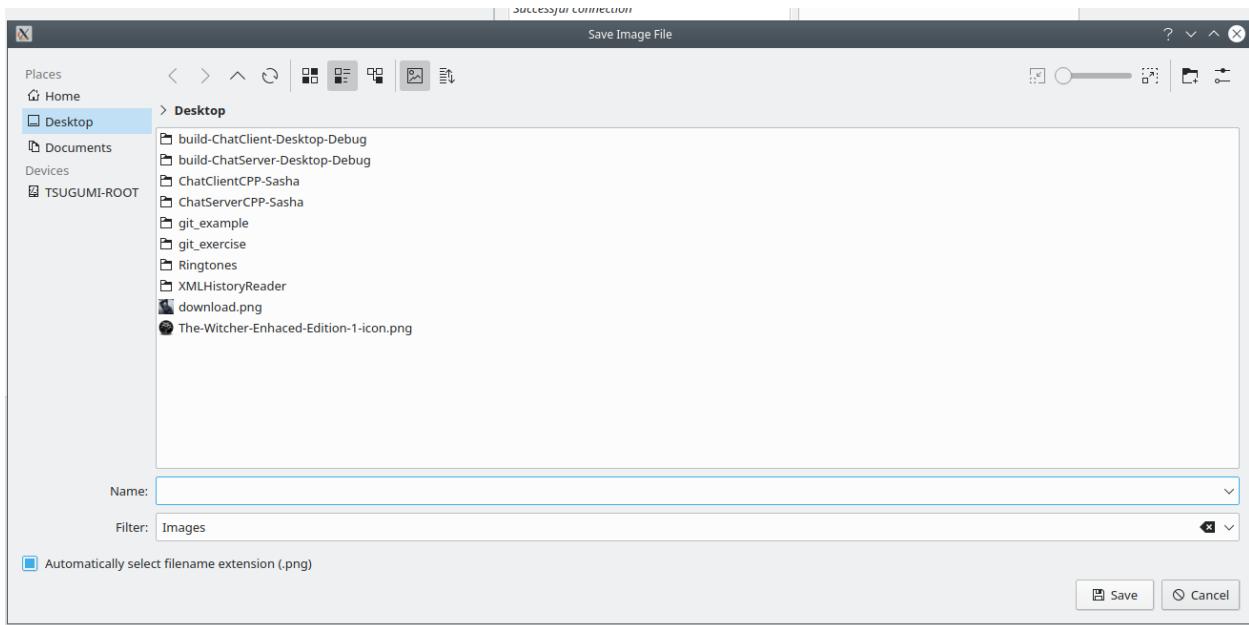


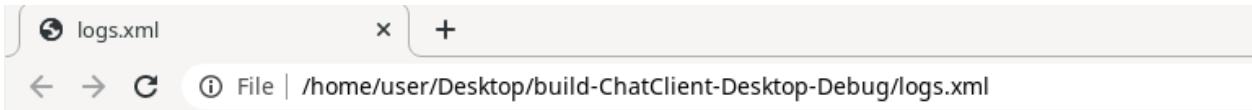
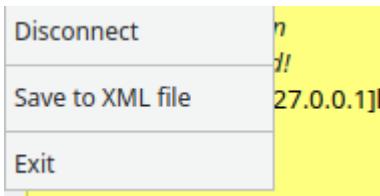






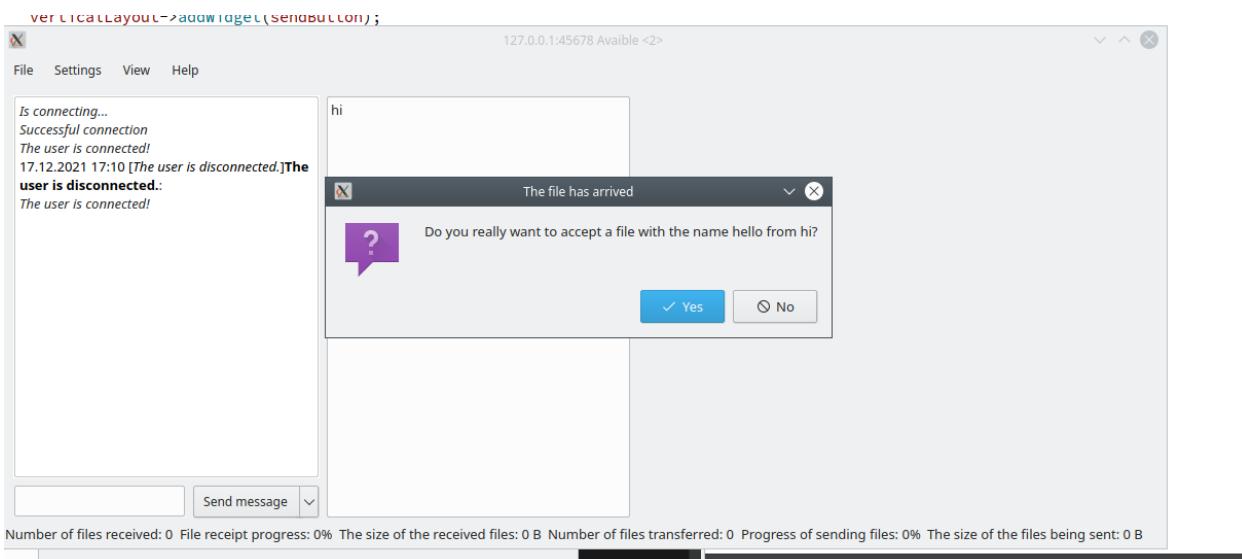
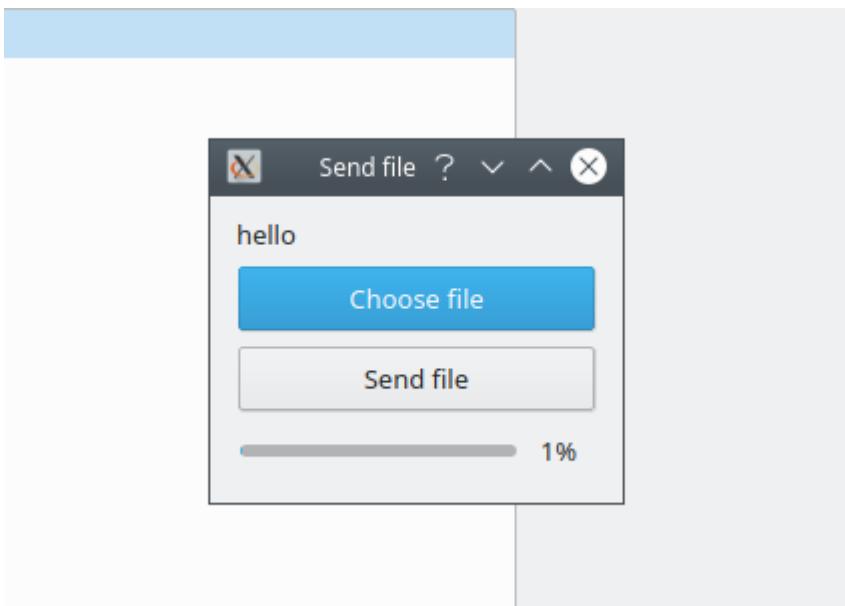
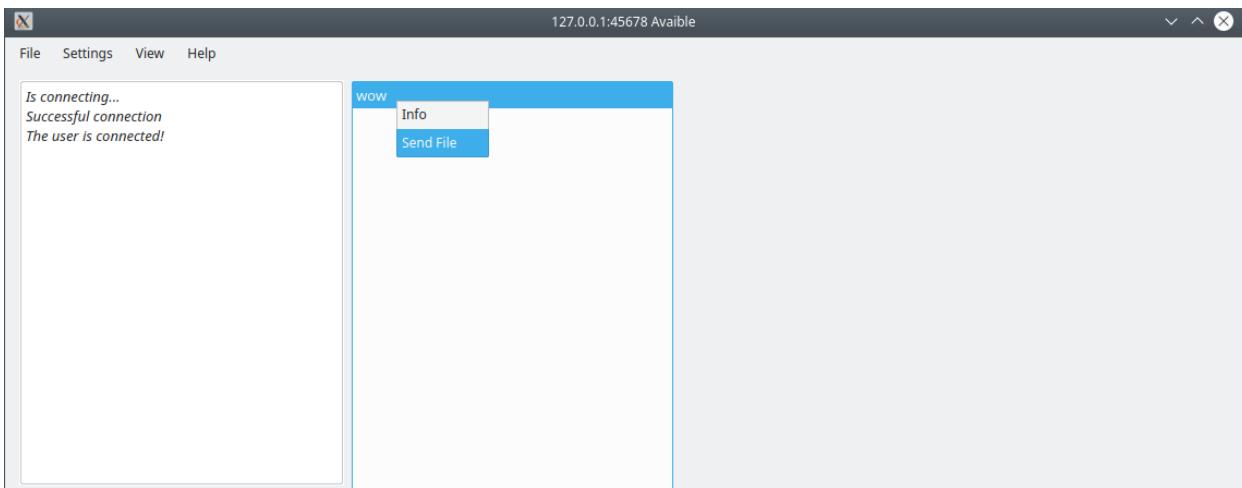




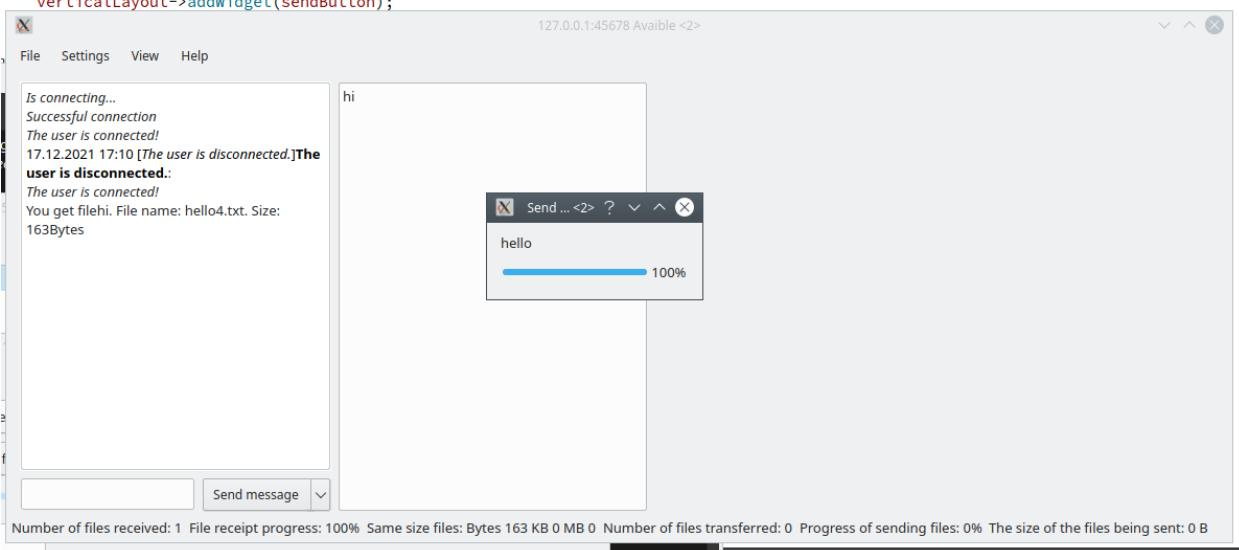


This XML file does not appear to have any style information associated with it. The document tree is shown below.

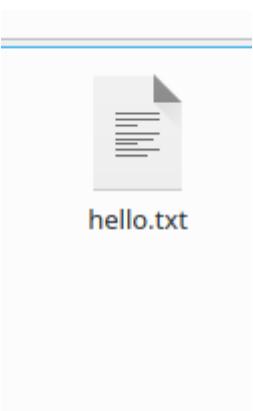
```
<?xml version="1.0" encoding="UTF-8"?>
<logs>
    <log number="1">
        <name>SERVER</name>
        <ip>127.0.0.1</ip>
        <dateAndTime>пт дек. 17 16:49:10 2021</dateAndTime>
        <serverMessage>The user is connected!</serverMessage>
    </log>
    <log number="2">
        <name>hi</name>
        <ip>127.0.0.1</ip>
        <dateAndTime>пт дек. 17 16:49:50 2021</dateAndTime>
        <message>jk</message>
    </log>
    <log number="3">
        <name>hi</name>
        <ip>127.0.0.1</ip>
        <dateAndTime>пт дек. 17 16:50:56 2021</dateAndTime>
        <imageMessage/>
    </log>
    <log number="4">
        <name>hi</name>
        <ip>127.0.0.1</ip>
        <dateAndTime>пт дек. 17 16:51:13 2021</dateAndTime>
        <imageMessage/>
    </log>
    <log number="5">
        <name>[127.0.0.1] </name>
        <ip>127.0.0.1</ip>
        <dateAndTime>пт дек. 17 16:51:25 2021</dateAndTime>
        <message>[127.0.0.1] </message>
    </log>
    <log number="6">
        <name>hi</name>
        <ip>127.0.0.1</ip>
        <dateAndTime>пт дек. 17 16:52:12 2021</dateAndTime>
        <imageMessage>iVBORw0KGgoAAAANSUhEUgAAAAUAAAAdwCAIAAAD+Tyo8AACXBIWXMAA7DAAA0wwHHb6hkAAAg</imageMessage>
    </log>
    <log number="7">
        <name>hi</name>
        <ip>127.0.0.1</ip>
        <dateAndTime>пт дек. 17 16:53:08 2021</dateAndTime>
    </log>
</logs>
```

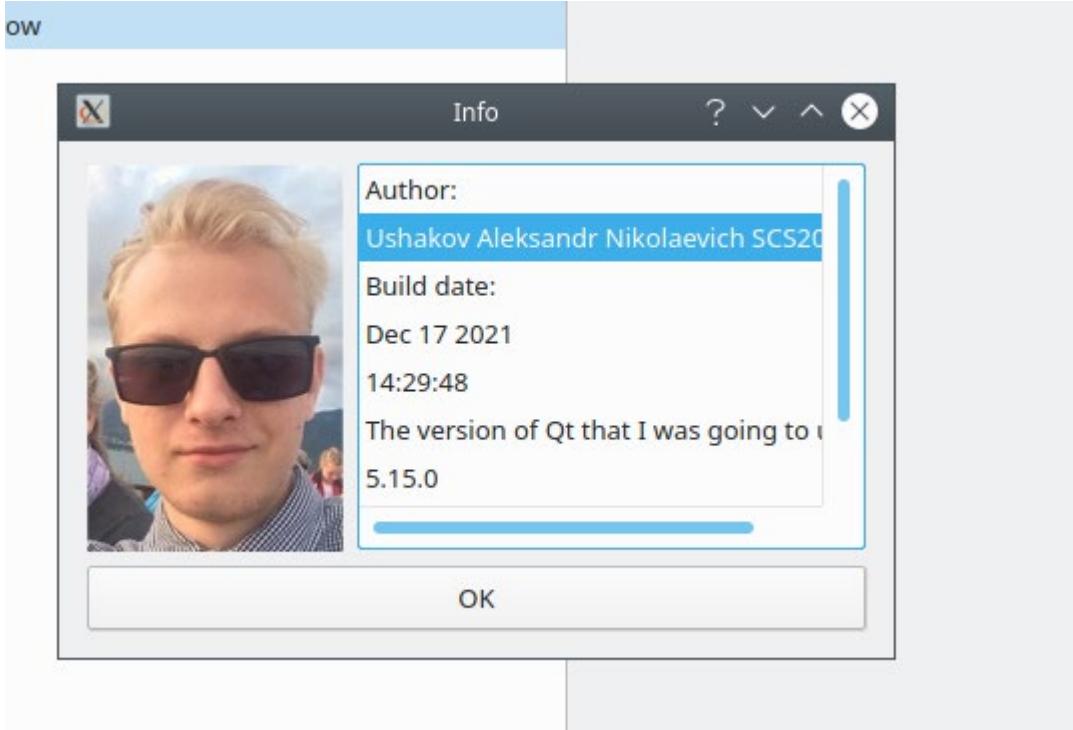


```
verticalLayout->addWidget(resolution);  
verticalLayout->addWidget(sendButton);
```



You get filehi. File name: hello4.txt. Size:
163Bytes





Приложение А

Листинг-1 A1.XML_Reader.pro

```
QT += xml
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000      # disables all the APIs
deprecated before Qt 6.0.0

SOURCES += \
```

```

main.cpp \
mainwindow.cpp

HEADERS += \
mainwindow.h

FORMS += \
mainwindow.ui

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

```

Листинг-2 А2.MainWindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QtWidgets/QMainWindow>
#include <QDomElement>
#include <QFile>
#include <QXmlStreamReader>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    //void testFunc(QDomElement element);

private slots:
    void on_action_Exit_triggered();
    void on_action_Exit_2_triggered();

private:
    Ui::MainWindow *ui;
    QDomDocument temp;
    // void extract(const QDomNode& a);

};


```

```
#endif // MAINWINDOW_HPP
```

Листинг-3 А3. MainWindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "QFileDialog"
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->action_Exit->setIcon(QIcon("C:\\Users\\The Witcher
Hunt\\Documents\\lab6XML_Reader\\choose.png"));
    ui->action_Exit_2->setIcon(QIcon("C:\\Users\\The Witcher
Hunt\\Documents\\lab6XML_Reader\\exit.jpg"));
    setWindowIcon(QIcon("C:\\Users\\The Witcher
Hunt\\Documents\\lab6XML_Reader\\xml.jpg"));
    setWindowTitle("XML Reader");

    QPalette darkPalette;

    // Настраиваем палитру для цветовых ролей элементов интерфейса
    darkPalette.setColor(QPalette::Window, QColor(53, 53, 53));
    darkPalette.setColor(QPalette::WindowText, Qt::black);
    darkPalette.setColor(QPalette::Base, QColor(25, 25, 25));
    darkPalette.setColor(QPalette::AlternateBase, QColor(53, 53,
53));
    darkPalette.setColor(QPalette::ToolTipBase, Qt::white);
    darkPalette.setColor(QPalette::ToolTipText, Qt::white);
    darkPalette.setColor(QPalette::Text, Qt::white);
    darkPalette.setColor(QPalette::Button, QColor(53, 53, 53));
    darkPalette.setColor(QPalette::ButtonText, Qt::white);
    darkPalette.setColor(QPalette::BrightText, Qt::red);
    darkPalette.setColor(QPalette::Link, QColor(42, 130, 218));
    darkPalette.setColor(QPalette::Highlight, QColor(42, 130,
218));
    darkPalette.setColor(QPalette::HighlightedText, Qt::black);

    // Устанавливаем данную палитру
    qApp->setPalette(darkPalette);

//    connect(ui->selectXMLButton, SIGNAL(clicked()), this,
//            SLOT(on_selectXMLButton_clicked()));
}

//void MainWindow::extract(const QDomNode& a) {
//    QDomNode t=a.firstChild();
//    while(!t.isNull()) {
//        if(t.isElement()) {
//            QDomElement h=t.toElement();
```

```

//           if(!h.isNull()) {
//               QString name;
//               QString text;
//               qDebug()<<h.text();
//               if(h.tagName()=="log") {
//                   QDomNode node=h.firstChild();
//                   /////
//                   /////
//                   //   QString name;
//                   //   QString text;
//                   while(!node.isNull()) {
//                       QDomElement e=node.toElement();
//                       if(!e.isNull()) {
//                           if(e.tagName() == "name") {
//                               name=e.text();
//                           } else if(e.tagName() == "message") {
//                               text=e.text();
//                           }
//                       }
//                       node=node.nextSibling();
//                   }
//                   /////
//                   /////
//                   /////
//                   /////
//                   /////
//                   ui->messageList->append(name);
//                   ui->messageList->append(text);
//                   qDebug()<<"F";
//                   ui->messageList->append(domElement.text());
//                   qDebug()<<"da";
//               }
//               ui->messageList->append(name);
//               ui->messageList->append(text);
//               qDebug()<<"F";
//           }
//       }
//   }
//   addFromXML(domNode);
//   t=t.nextSibling();
// }
// }

MainWindow::~MainWindow() {
    delete ui;
}

void MainWindow::on_action_Exit_triggered()
{
    //Выбор файла, диалговое окно, потом создаем QDomElement и т.п.
    //Сохраняем его в поле класса и вызываем отдельный метод выгрузки
    //данных в QTextEdit
    QString a = QFileDialog::getOpenFileName(this, tr("Open XML"),
    "C:\\\\Users\\\\The Witcher Hunt\\\\Desktop", tr("XML Files (*.xml)"));
    if (!a.isEmpty()){
        QFile file(a);
        if(file.open(QIODevice::ReadOnly)) {
            if(temp.setContent(&file)) {

```

```

//                                     QDomElement domElement=domDoc.documentElement();
//                                     addFromXML(domElement);
QDomElement topElement=temp.documentElement();
QDomNode domNode=topElement.firstChild();
while(!domNode.isNull()) {
    QDomElement domElement=domNode.toElement();
    if(domElement.tagName() == "log") {
        QDomNode node=domElement.firstChild();
        while(!node.isNull()) {
            QDomElement element=node.toElement();
            if(!element.isNull()) {
                QString log;
                QString tagName(element.tagName());
                if(tagName=="name") {

log.append("<strong>" + element.text() + "</strong>: ");

if(node.nextSibling().tagName() == "ip") {
    log.prepend("[ " + node.nextSibling().text() + " ] ");
}

if(node.nextSibling().nextSiblingElement().tagName() == "dateAndTime") {
    log.prepend(node.nextSibling().nextSiblingElement().text());
}

if(node.nextSibling().nextSibling().nextSiblingElement().tagName() == "message") {
    log.append(node.nextSibling().nextSibling().nextSiblingElement().text());
    ui->messageList->append(log);
} else
if(node.nextSibling().nextSibling().nextSiblingElement().tagName() == "serverMessage") {

log.append("<em>" + node.nextSibling().nextSibling().nextSiblingElement().text() + "</em>");
    ui->messageList->append(log);
} else
if(node.nextSibling().nextSibling().nextSiblingElement().tagName() == "formattedMessage") {
    QString
message=node.nextSibling().nextSibling().nextSiblingElement().text().mid(
node.nextSibling().nextSibling().nextSiblingElement().text().indexOf(")") + 1);
    QString
color=node.nextSibling().nextSibling().nextSiblingElement().text().mid

```

```

(node.nextSibling().nextSibling().nextSiblingElement().text().indexOf(
"(")+1,
node.nextSibling().nextSibling().nextSiblingElement().text().indexOf(",
")-1);

                QString
font=node.nextSibling().nextSibling().nextSiblingElement().text().mid(
node.nextSibling().nextSibling().nextSiblingElement().text().indexOf(",
")+1, node.nextSibling().nextSiblingElement().text().indexOf(") ") -2);

                QString
weight=node.nextSibling().nextSibling().nextSiblingElement().text().mi-
d(node.nextSibling().nextSiblingElement().text().indexOf(") ") -2,
node.nextSibling().nextSibling().nextSiblingElement().text().indexof(")"));

log.append("<font
color="+color/*+" font-size='"+weight+" font-
family='"+font+*/+">" +message+"</font>");
ui->messageList->append(log);
} else
if(node.nextSibling().nextSibling().nextSiblingElement().tagName() == "i-
mageMessage") {
ui->messageList->append(log);
QImage
image=QImage::fromData(QByteArray::fromBase64(node.nextSibling().nextS-
ibling().nextSiblingElement().text().toUtf8()), "PNG");
QImage
image_new=image.scaled(320,240);
QTextCursor newCursor=ui-
>messageList->textCursor();

newCursor.movePosition(QTextCursor::End);
ui->messageList-
>setTextCursor(newCursor);
ui->messageList-
>textCursor().insertBlock();
ui->messageList-
>textCursor().insertImage(image_new);
} else
if(node.nextSibling().nextSibling().nextSiblingElement().tagName() == "f-
ileMessage") {
QString filename;
QString md5;
//filename=node.nextSibling().nextSibling().nextSiblingElement().text().left(
node.nextSibling().nextSibling().nextSiblingElement().text().inde-
xOf(","));
//md5=node.nextSibling().nextSibling().nextSiblingElement().text().right

```

```

(node.nextSibling().nextSibling().nextSiblingElement().text().indexOf(
",") + 1);

log.append(node.nextSibling().nextSibling().nextSiblingElement().text());
ui->messageList->append(log);
}
} else if(tagName=="message") {
} else if(tagName=="serverMessage") {

}
node=node.nextSibling();
}
domNode=domNode.nextSibling();
}
file.close();}

// QDomElement root=domDoc.documentElement();
// QDomNode node=root.firstChild();
// while(!node.isNull()) {
//     if(node.toElement().tagName() == "logs") {
//         testFunc(node.toElement());
//         node=node.nextSibling();
//     }
// }

// QDomNodeList logs=root.elementsByTagName("log");
// for(std::size_t i=0; i<logs.count(); ++i) {
//     QDomNode log=logs.at(i);
//     if(log.isElement()) {
//         QDomElement message=log.toElement();
//         qDebug() << "In main:" << message.attribute("name");
//         testFunc(message, "log", "name");
//     }
// }
}

void MainWindow::on_action_Exit_2_triggered()
{
    this->close();
}

```

Листинг-4 A4.main.cpp

```
#include "mainwindow.h"
```

```

#include <QApplication>
#include <QStyleFactory>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    qApp->setStyle(QStyleFactory::create("Fusion"));

    MainWindow w;
    w.resize(600, 410);
    w.show();
    return a.exec();
}

```

Приложение Б

Листинг-5 Server_Settings.h

```

#ifndef SERVER_SETTINGS_H
#define SERVER_SETTINGS_H
#include <QString>
#include <QLineEdit>
#include <QPushButton>
#include <QLabel>
#include <QWidget>
#include <QDialog>
#include <QLayout>
#include <QMMessageBox>
#include <QtNetwork/QTcpServer>
#include <QMainWindow>

class Server_Settings : public QDialog {
    Q_OBJECT

public:
    Server_Settings(QWidget *parent=nullptr);
    ~Server_Settings();
    QPushButton *accept;
    QLineEdit *ip;
    QLineEdit *port;

private:
    QLabel *ipLabel;
    QLabel *portLabel;
    QHBoxLayout *ipHorizontalLayout;
    QHBoxLayout *portHorizontalLayout;
}

```

```
QVBoxLayout *verticalLayout;  
};  
#endif // SERVER_SETTINGS_H
```

Листинг-6 Б2. Server_Settings.cpp

```
#include "Server_Settings.h"  
#include <QHBoxLayout>  
  
Server_Settings::Server_Settings(QWidget *parent) : QDialog(parent) {  
    ipLabel=new QLabel(tr("IP: "));  
    portLabel=new QLabel(tr("Port: "));  
    ip=new QLineEdit();  
    //    ipLineEdit->setText(server->getIp());  
    port=new QLineEdit();  
    //    portLineEdit->setText(server->getPort());  
    accept=new QPushButton(tr("&Accept"));  
    //    MainWindow Server(parent);  
    //    server=&Server;  
  
    // чтобы редактор оставался активен, пока открыто окно  
    setFocusProxy(ip);  
  
    // устанавливаем все виджеты и слои  
    ipHorizontalLayout=new QHBoxLayout();  
    portHorizontalLayout=new QHBoxLayout();  
    verticalLayout=new QVBoxLayout();  
    verticalLayout->addLayout(ipHorizontalLayout);  
    verticalLayout->addLayout(portHorizontalLayout);  
    verticalLayout->addWidget(accept);  
    ipHorizontalLayout->addWidget(ipLabel);  
    ipHorizontalLayout->addWidget(ip);  
    portHorizontalLayout->addWidget(portLabel);  
    portHorizontalLayout->addWidget(port);  
    setLayout(verticalLayout);  
    setWindowTitle(tr("Server settings"));  
    connect(accept, SIGNAL(clicked()), parent, SLOT(accept()));  
}  
  
Server_Settings::~Server_Settings() {  
    delete ipLabel;  
    delete portLabel;  
    delete ip;  
    delete port;  
    delete accept;  
    delete ipHorizontalLayout;  
    delete portHorizontalLayout;  
    delete verticalLayout;
```

```
}
```

```
//void Server_Settings::on_acceptButton_clicked() {
//    QString ips=ip->text();
//    QString ports=port->text();

//    if (ips.isEmpty()||ports.isEmpty()) {
//        QMessageBox::information(this, tr("Empty line"), tr("Repeat your
action"));
//        return;
//    }
//    //    server->setIp(ip);
//    //    server->setPort(port);
//    //    server->setTitleIpPort();

//}
```

Листинг-7 user.h

```
#ifndef USER_H
#define USER_H

#include <QByteArray>
#include <QVector>
#include <QString>
#include <QtNetwork/QTcpSocket>
#include <QIcon>
#include <QDateTime>

class User {
public:
    User();
    QString username="User";
    QTcpSocket user_socket;
    QString user_status="Avaible";
    QImage user_icon;
    QString ip="127.0.0.1";
    QByteArray user_icon_64;
    bool autoFileGet=0;
    QDateTime connected_time;

private:
    //    QString username="Test";
    QByteArray user_id;
    QVector<QByteArray> current_users;

public:
    void add_user(QByteArray);
    void remove_user(QByteArray);
```

```

    bool find_user(QByteArray);
    void clear_users();
    void set_user_id(QByteArray);
    void set_username(QString);
    void set_user_icon(QImage);
    QByteArray get_user_id();
    QString get_user_name();
};

#endif // USER_H

```

Листинг-8 user.cpp

```

#include "user.h"

User::User ()=default;

void User::add_user(QByteArray contact) {
    current_users.push_back(contact);
}

void User::remove_user(QByteArray contact) {
    for(std::size_t i=0; i<current_users.size(); ++i) {
        if(current_users[i]==contact) {
            current_users.remove(i);
        }
    }
}

bool User::find_user(QByteArray contact) {
    for(std::size_t i=0; i<current_users.size(); ++i) {
        if(current_users[i]==contact) {
            return true;
        }
    }
    return false;
}

void User::clear_users() {
    current_users.clear();
}

void User::set_user_id(QByteArray user_id) {
    this->user_id=user_id;
}

void User::set_username(QString name) {
    this->username=name;
}

void User::set_user_icon(QImage image) {
    this->user_icon=image;
}

```

```

}

QByteArray User::get_user_id() {
    return user_id;
}

QString User::get_user_name() {
    return username;
}

```

Листинг-9 mainwindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

#include <QtWidgets>
#include <QMainWindow>
#include <QtNetwork/QTcpServer>
#include <QtNetwork/QTcpSocket>
#include <QPlainTextEdit>
#include <QHostAddress>
#include "Server_Settings.h"
#include <QSql>
#include <QDomDocument>
#include "user.h"

class MainWindow : public QWidget {

    Q_OBJECT

public:
    MainWindow();
    void sendMessage(const QString &message);

public slots:
    void accept();
private slots:
    void newConnection();
    void dataReceived();
    void logoutUser();
    void start();
    void stop();
    void on_saveLogs();
    void on_network();

private:
    void createMenu();
    QPushButton *testButton;
    QMenu *fileMenu;
}

```

```

QMenu *settingsMenu;
QAction
*networkAction,*startAction,*stopAction,*exitAction,*saveXMLAction;
QTextEdit *logs;
QLabel *serverPos;

quint16 port=45678;
QHostAddress ip=QHostAddress("127.0.0.1");
QTcpServer *server;
 QList<QTcpSocket *> users;
 QMap<QTcpSocket *, User *> userInfo;
quint16 messageLength;
Server_Settings *sDialog;
QByteArray currentByte;
QDomDocument *doc;
QDomElement domElement;
void userListUpdated();
void saveXMLFile();
QDomElement log(QDomDocument *domDoc, const QString &dateTime, const
QString &ip, const QString &name, const QString &type, const QString
&message);
    QDomElement makeElement(QDomDocument *domDoc, const QString &strName,
const QString &stdAttr, const QString &Text);
    QMap<QString, QList<QString>> filesList;

};

#endif // MAINWINDOW_H

```

Листинг-10 mainwindow.cpp

```

#include "mainwindow.h"
#include "QMenuBar"

MainWindow::MainWindow() {
    QPalette darkPalette;

    // Настраиваем палитру для цветовых ролей элементов интерфейса
    darkPalette.setColor(QPalette::Window, QColor(53, 53, 53));
    darkPalette.setColor(QPalette::WindowText, Qt::black);
    darkPalette.setColor(QPalette::Base, QColor(25, 25, 25));
    darkPalette.setColor(QPalette::AlternateBase, QColor(53, 53,
53));
    darkPalette.setColor(QPalette::ToolTipBase, Qt::white);
    darkPalette.setColor(QPalette::ToolTipText, Qt::white);
    darkPalette.setColor(QPalette::Text, Qt::white);
    darkPalette.setColor(QPalette::Button, QColor(53, 53, 53));
    darkPalette.setColor(QPalette::ButtonText, Qt::white);
    darkPalette.setColor(QPalette::BrightText, Qt::red);
    darkPalette.setColor(QPalette::Link, QColor(42, 130, 218));
}

```

```

    darkPalette.setColor(QPalette::Highlight, QColor(42, 130,
218));
    darkPalette.setColor(QPalette::HighlightedText, Qt::black);

    // Устанавливаем данную палитру
    qApp->setPalette(darkPalette);
serverPos = new QLabel;
logs=new QTextEdit();
logs->setReadOnly(true);
logs->acceptRichText();
createMenu();

doc=new QDomDocument("logs");
domElement=doc->createElement("logs");
doc->appendChild(domElement);

sDialog=new Server_Settings(this);

QHBoxLayout *mainLayout=new QHBoxLayout;
QVBoxLayout *tabLayout=new QVBoxLayout;
QVBoxLayout *layout = new QVBoxLayout;
layout->addWidget(serverPos);
layout->addWidget(logs);
//layout->addWidget(quitButton);
mainLayout->addLayout(layout);
mainLayout->addLayout(tabLayout);
this->setLayout(mainLayout);
QString temp="127.0.0.1";
setWindowTitle("Server "+temp+" "+QString::number(port)+" 0");

//      testButton=new QPushButton("Tect");
//      testButton-
>setIcon(QIcon(QPixmap::fromImage(QImage ("/home/user/Desktop/photo.jpg
"))));
//      mainLayout->addWidget(testButton);
messageLength = 0;

}

void MainWindow::on_saveLogs() {
saveXMLFile();
}

void MainWindow::createMenu() {
//      MenuFile = menuBar()->addMenu ("Файл");
//      MenuEdit = menuBar()->addMenu ("Правка");
//      MenuFormat=menuBar()->addMenu ("Формат");
//      MenuView=menuBar()->addMenu ("Вид");
//      MenuHelp=menuBar()->addMenu ("Справка");
}

```

```

//      New=MenuFile->addAction("Новый файл");
//      Open=MenuFile->addAction("Открыть");
//      Save=MenuFile->addAction("Сохранить");
//      Save_as=MenuFile->addAction("Сохранить как");
//      Exit=MenuFile->addAction("Выход");
QMenuBar *t=new QMenuBar(this);
fileMenu=t->addMenu("&File");
settingsMenu=t->addMenu("&Settings");
networkAction=new QAction(tr("Сеть"));
startAction=new QAction(tr("On"));
stopAction=new QAction(tr("Off"));
exitAction=new QAction(tr("Quit"));
saveXMLAction=new QAction(tr("Save to XML file"));
fileMenu->addAction(startAction);
fileMenu->addAction(stopAction);
fileMenu->addAction(saveXMLAction);
fileMenu->addAction(exitAction);
settingsMenu->addAction(networkAction);
QMenuBar *menu=new QMenuBar(this);
menu->addMenu(fileMenu);
menu->addMenu(settingsMenu);
connect(exitAction, SIGNAL(triggered()), qApp, SLOT(quit()));
connect(startAction, SIGNAL(triggered()), this, SLOT(start()));
connect(stopAction, SIGNAL(triggered()), this, SLOT(stop()));
connect(saveXMLAction, SIGNAL(triggered()), this,
SLOT(on_saveLogs()));
connect(networkAction, SIGNAL(triggered()), this,
SLOT(on_network()));
}

void MainWindow::start() {
    server = new QTcpServer(this);
    if(!server->listen(ip, port)) {
        serverPos->setText(tr("The server could not start: ") +
server->errorString());
        logs->append(tr("The server could not start: ") +server-
>errorString());
    } else {
        setWindowTitle("IP: "+ip.toString()+" PORT:
"+QString::number(port)+" 0");
//        serverState->setText("\n ");
//        serverState->setText(" ");
//        serverState->setText(" ");
//        serverState->setText(tr("\nServer is exist on port ") +
QString::number(server->serverPort()));
        logs->append(tr("Server started! "));
        connect(server, SIGNAL(newConnection()), this,
SLOT(newConnection()));
    }
}

```

```

void MainWindow::stop() {
    logs->append(tr("Server has been stopped!"));
    for(auto user : users) {
        user->disconnectFromHost();
        usersInfo.remove(user);
        users.removeOne(user);
    }

    server->disconnect();
    server->close();
    setWindowTitle("Server "+QString::number(port)+" "+ip.toString());
}

void MainWindow::on_network() {
    sDialog->ip->setText(ip.toString());
    sDialog->port->setText(QString::number(port));
    sDialog->open();
}

void MainWindow::accept() {
    if (sDialog->ip->text().isEmpty() || sDialog->port-
>text().isEmpty()) {
        QMessageBox::information(sDialog, tr("Empty line"), tr("Repeat
action"));
        return;
    }
    ip=QHostAddress(sDialog->ip->text());
    port=sDialog->port->text().toUInt();
    setWindowTitle("Server "+ip.toString()+" "+QString::number(port)+"
0");
    //QMessageBox::information(sDialog, tr("Успех!"), tr("Вы успешно
поменяли настройки сервера!"));
    sDialog->hide();
}

void MainWindow::userListUpdated() {
    QString names;
    QByteArrayList images_64;
    QStringList ips;
    QStringList times;
    QStringList statuses;
    for(std::size_t i=0; i<usersInfo.count(); ++i) {
        names.append(usersInfo.value(users[i])>username);
        names.append(", ");
        images_64<<usersInfo.value(users[i])>user_icon_64;
        ips<<usersInfo.value(users[i])>ip;
        statuses<<usersInfo.value(users[i])>user_status;
        times<<usersInfo.value(users[i])>connected_time.toString();
    }
    qDebug()<<"Список пользователей онлайн: "<<names;
    qDebug()<<"Список их картинок: "<<images_64;
}

```

```

quint16 amount=images_64.count();
for(std::size_t i=0; i<usersInfo.count(); ++i) {
    QByteArray block;
    QDataStream out(&block, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_1);

out<<(quint16) 0<<(QString) "USERLIST_UPDATED"<<names<<images_64<<ips<<times<<statuses;
//           qDebug() <<"IMAGES"<<images_64;
    users[i]->write(block);
    users[i]->flush();
}
}

void MainWindow::newConnection() {

QTcpSocket* newUser = server->nextPendingConnection();
users << newUser;
User *user=new User();
usersInfo.insert(newUser, user);

connect(newUser, SIGNAL(readyRead()), this, SLOT(dataReceived()));
connect(newUser, SIGNAL(disconnected()), this,
SLOT(logoutUser()));

QString message="User is connected!";
setWindowTitle(ip.toString() + ":" + QString::number(port) +
"+QString::number(users.count()));
QDomElement newLog=log(doc,
QDateTime::currentDateTime().toString(), newUser-
>peerAddress().toString(), "SERVER", "serverMessage", message);
domElement.appendChild(newLog);

sendMessage("<em>User is connected!</em>");
}

void MainWindow::dataReceived() {
int mainData=0;
//packet received
//Whose packet is it?

QTcpSocket *socket = qobject_cast<QTcpSocket *>(sender());
if(socket == 0) { //can't find sender
    return;
}
//otherwise lets continue
//Getting message
QDataStream in(socket);
in.setVersion(QDataStream::Qt_4_1);
//      in.setVersion(QDataStream::Qt_5_5);
if(messageLength == 0) {

```

```

if(socket->bytesAvailable() < (int) sizeof(quint16)) {
    return;
}
in >> messageLength;
}
//did we get all the bytes?
if(socket->bytesAvailable() < messageLength) {

    return; //no
}
//yes

if(messageLength==0) {
    //TODO Картинку сохранять и добавлять в логи.
    //Файлу генерировать стремное название и сохранять в Downloads
    //-----SEND PHOTO
    QString test;
    QString username;
    QByteArray icon;
    QString name;
    QString status;
    QByteArray icon_64;
    in>>test;
    if(test=="ICON") {
        in>>username;
        in>>icon;
    } else if(test=="USERLIST_UPDATE") {
        in>>name;
        in>>icon_64;
        in>>status;
    }

    if(test=="ICON") {
        QImage
image=QImage::fromData(QByteArray::fromBase64(icon), "PNG");
        //QLabel *label=new QLabel(this);
        //label->setPixmap(QPixmap::fromImage(image));
        //label->show();
        testButton->setIcon(QIcon(QPixmap::fromImage(image)));
        usersInfo.value(socket)->username=username;
        usersInfo.value(socket)->set_user_icon(image);
        usersInfo.value(socket)->user_icon_64=icon;
        messageLength=0;

        QBuffer buffer;
        QByteArray block;
        QDataStream out(&block, QIODevice::WriteOnly);
        out.setVersion(QDataStream::Qt_4_1);
        buffer.open(QIODevice::WriteOnly);

```

```

        image.save(&buffer, "PNG");
        QString encoded=buffer.data().toBase64();

out<<(quint16)0<<(QString)"UPDATED_USER_ICON"<<username<<icon;//encode
d;

        for(int i = 0; i < users.size(); ++i) {
            if(users[i]!=socket) {
                users[i]->write(block);
                users[i]->flush();
            }
        }

mainData=1;
messageLength=0;
} else if(test=="FILESENDED") {
    QByteArray block;
    QString filename;
    QString username;
    in>>username;
    in>>filename;
    in>>block;

    for(std::size_t i=0; i<users.count(); ++i) {
        if(usersInfo.value(users[i])->username==username) {
            QByteArray packet;
            QDataStream out(&packet, QIODevice::WriteOnly);
            out.setVersion(QDataStream::Qt_4_1);

out<<(quint16)0<<(QString)"FILEFORYOU"<<filename<<block<<usersInfo.value(socket)->username;
            users[i]->write(packet);

            users[i]->flush();
        }
    }
    QByteArray packetForFirstClient;
    QDataStream out(&packetForFirstClient,
QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_1);
    out<<(quint16)0<<(QString)"FILESENDEDC";
    socket->write(packetForFirstClient);
    socket->flush();
    messageLength=0;
    mainData=1;
}

if(QFileInfo("/home/user/DownloadsFromChat/"+filename+".txt").exists())
{
//          int i=1;
//          bool numberFound=false;
}

```

```

//           while(numberFound==false) {
//           QString filename_copy=filename;
//           filename_copy.append(QString::number(i)+".txt");
//           }
//           if(!QFileInfo("/home/user/DownloadsFromChat/"+filename_copy).exists())
//           {
//               numberFound=true;
//               filename=filename_copy;
//               break;
//           }
//           ++i;
//           }
//           QFile
file("/home/user/DownloadsFromChat/"+filename);
//           if(file.open(QIODevice::ReadWrite)) {
//           QTextStream stream(&file);
//           stream<<block;
//           file.close();

//           QByteArray blockForUser;
//           QDataStream out(&blockForUser,
QIODevice::WriteOnly);
//           out.setVersion(QDataStream::Qt_4_1);
//           out<<(quint16)0<<(QString)"FILESENDED";
socket->write(blockForUser);
messageLength=0;
mainData=1;
//           }
//           } else {
//           filename.append(".txt");
//           QFile
file("/home/user/DownloadsFromChat/"+filename);
//           if(file.open(QIODevice::ReadWrite)) {
//           QTextStream stream(&file);
//           stream<<block;
//           file.close();

//           QByteArray blockForUser;
//           QDataStream out(&blockForUser,
QIODevice::WriteOnly);
//           out.setVersion(QDataStream::Qt_4_1);
//           out<<(quint16)0<<(QString)"FILESENDED";
socket->write(blockForUser);
messageLength=0;
mainData=1;
//           }
//           }
} else if(test=="FILEGETACCEPT") {
QString usernameGet=usersInfo.value(socket)->username;
QString usernameSend=filesList.value(usernameGet)[0];
QString filename=filesList.value(usernameGet)[1];

```

```

    quint16 size=filesList.value(usernameGet)[2].toUInt();
    QByteArray
fileBlock=QByteArray::fromBase64(filesList.value(usernameGet)[3].toUtf
8());;

    QByteArray block;
    QDataStream out(&block, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_1);

out<<(quint16) 0<<(QString) "FILEFORCLIENT"<<usernameSend<<filename<<siz
e<<fileBlock;

    socket->write(block);
//    socket->flush();

    QByteArray blockForFirstClient;
    QDataStream out1(&blockForFirstClient,
QIODevice::WriteOnly);
    out1.setVersion(QDataStream::Qt_4_1);
    out1<<(quint16) 0<<(QString) "CLIENTACCEPTFILE";
    for(std::size_t i=0; i<usersInfo.count(); ++i) {
        if(usersInfo.value(users[i])->username==usernameSend)
{
            users[i]->write(blockForFirstClient);

            users[i]->flush();
        }
    }
    filesList.remove(usernameGet);
    messageLength=0;
    mainData=1;
    return;
} else if(test=="FILEGETDECLINE") {
    QString usernameGet=usersInfo.value(socket)->username;
    QString usernameSend=filesList.value(usernameGet)[0];

    QByteArray blockForFirstClient;
    QDataStream out1(&blockForFirstClient,
QIODevice::WriteOnly);
    out1.setVersion(QDataStream::Qt_4_1);
    out1<<(quint16) 0<<(QString) "CLIENTDECLINEFILE";
    for(std::size_t i=0; i<usersInfo.count(); ++i) {
        if(usersInfo.value(users[i])->username==usernameSend)
{
            users[i]->write(blockForFirstClient);
            users[i]->flush();
        }
    }
    filesList.remove(usernameGet);
    messageLength=0;
}

```

```

        mainData=1;
    //        socket->flush();
} else if(test=="FILEFORSERVER") {
    QString username;
    QString filename;
    quint16 size;
    QByteArray block;
    in>>username;
    in>>filename;
    in>>size;
    in>>block;

    QByteArray packetForFirstClient;
    QDataStream out(&packetForFirstClient,
QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_1);
    out<<(quint16)0<<(QString)"SERVERGOTFILE";
//    socket->write(packetForFirstClient);
//    socket->flush();

    QList<QString> list;
    list.append(userInfo.value(socket)->username);
    list.append(filename);
    list.append(QString::number(size));
    list.append(QString::fromUtf8(block.toBase64()));
    filesList.insert(username, list);

    for(std::size_t i=0; i<users.count(); ++i) {
        if(userInfo.value(users[i])->username==username) {
//            QByteArray secondPacketForFirstClient;
//            QDataStream out2(&secondPacketForFirstClient,
QIODevice::WriteOnly);
//            out2.setVersion(QDataStream::Qt_4_1);
//            out2<<(quint16)0<<(QString)"CLIENTASKEDFORFILE";
//            socket->write(secondPacketForFirstClient);
//            socket->flush();

            QByteArray packet;
            QDataStream out(&packet, QIODevice::WriteOnly);
            out.setVersion(QDataStream::Qt_4_1);
            //DELETED "<<block<<

out<<(quint16)0<<(QString)"ASKFORFILEGET"<<filename<<userInfo.value(socket)->username;
            users[i]->write(packet);
            users[i]->flush();
        }
    }

messageLength=0;

```

```

        mainData=1;
        return;

    } else if(test=="USERLIST_UPDATE") {
        User *newUser=new User();
        newUser->username=name;
        newUser->user_icon_64=icon_64;
        newUser->user_status=status;
        newUser->ip=socket->peerAddress().toString();
        newUser->connected_time=QDateTime::currentDateTime();
        usersInfo.insert(socket, newUser);
        userListUpdated();
        messageLength=0;
        mainData=1;
    } else if(test=="USER_UPDATED") {
        User *newUser=new User();
        QString username;
        QByteArray icon_64;
        QString status;
        in>>username;
        in>>icon_64;
        in>>status;
        QDateTime oldTime;
        if(usersInfo.contains(socket)) {
            oldTime=usersInfo.take(socket)->connected_time;
        }
        newUser->username=username;
        newUser->user_icon_64=icon_64;
        newUser->user_status=status;
        newUser->connected_time=oldTime;
        newUser->ip=socket->peerAddress().toString();
        usersInfo.insert(socket, newUser);
        userListUpdated();
        messageLength=0;
        mainData=1;
    } else if(test=="PHOTO") {
        QByteArray image_64;
        QString author;
        in>>author;
        in>>image_64;

        /*QImage
image=QImage::fromData(QByteArray::fromBase64(image_64), "PNG");
QBuffer buffer;
buffer.open(QIODevice::WriteOnly);
image.save(&buffer, "PNG");
QString encoded=buffer.data().toBase64();
QDomElement newLog=log(doc,
 QDateTime::currentDateTime().toString(), socket-
>peerAddress().toString(), author, "imageMessage", encoded);
domElement.appendChild(newLog);*/

```

```

QString ipString=usersInfo.value(socket)->ip;
QByteArray block;
QDataStream out(&block, QIODevice::WriteOnly);
out.setVersion(QDataStream::Qt_4_1);

out<<(quint16)0<<(QString)"PHOTO"<<author<<ipString<<image_64;
    for(std::size_t i=0; i<users.count(); ++i) {
        users[i]->write(block);
        users[i]->flush();
    }
    messageLength=0;
    mainData=1;
} else if(test=="FORMATTED_MESSAGE") {
    QString fontName;
    int fontWeight;
    QString colorStr;
    QString username;
    QString message;
    in>>username;
    in>>colorStr;
    in>>fontName;
    in>>fontWeight;
    in>>message;
    QByteArray packet;
    QDataStream out(&packet, QIODevice::WriteOnly);

    QString
messageNew="("+colorStr+", "+fontName+", "+QString::number(fontWeight)+"
) "+message;
    QDomElement newLog=log(doc,
QDateTime::currentDateTime().toString(), socket-
>peerAddress().toString(), username, "formattedMessage", messageNew);
    domElement.appendChild(newLog);

out<<(quint16)0<<(QString)"FORMATTED_MESSAGE"<<username<<colorStr<<fon
tName<<fontWeight<<(QString)socket->peerAddress().toString()<<message;
    for(std::size_t i=0; i<users.count(); ++i) {
        users[i]->write(packet);
        users[i]->flush();
    }
    messageLength=0;
    mainData=1;
} else if(test=="FILE") {
    //Сохраняем файл с прикольным названием
} else if(test=="USERNAME_CHANGE") {
    QString oldUsername;
    QString newUsername;
    in>>oldUsername;
    in>>newUsername;
}

```

```

        for(auto user : usersInfo) {
            if(user->username==oldUsername) {
                QTcpSocket *savedSocket=usersInfo.key(user);
                QByteArray
            savedImage=usersInfo.value(savedSocket)->user_icon_64;
                User *savedUser=new User();
                savedUser->username=newUsername;
                savedUser->user_icon_64=savedImage;
                usersInfo.remove(usersInfo.key(user));
                usersInfo.insert(savedSocket, savedUser);
            }
        }
        userListUpdated();
        mainData=1;
        messageLength=0;
    } else if(test=="DISCONNECT") {
        QString username;
        in>>username;
        // usersInfo.remove(socket);
        // users.removeOne(socket);
        // userListUpdated();
    } else {

    }
}
QString message;
in >> message;

logs->append(message);

if(message.contains("[USERNAME]")) {
    int needPos=message.indexOf("]");
    int needPos2=message.indexOf(",");
    int needPos21=message.indexOf("]", needPos2);
    int needPos3=message.indexOf(",", needPos2+1);
    int needPos4=message.indexOf("]", needPos3);
    int needPos41=message.indexOf(",", needPos4);
    int needPos5=message.indexOf("]", needPos41);
    // int needPos51=message.indexOf()
    QString name=message.mid(needPos+2, needPos2-needPos-2);
    QString status=message.mid(needPos21+1, needPos3-needPos21-
1/*message.count()-1*/);
    QString autoFile=message.mid(needPos4+1, needPos41-needPos4-
1);
    // QString icon_bits=message.mid(needPos5+1, message.count()-
needPos5-1);
    QByteArray icon_bits=message.mid(needPos5+1, message.count()-
needPos5-1).toUtf8().toBase64();
    QPixmap image;
}

```

```

image.loadFromData(QByteArray::fromBase64(icon_bits));
QIcon icon;
if(usersInfo.contains(socket)) {
    for(auto user : users) {
        if(socket==user) {
            User *newUser=new User();
            newUser->set_username(name);
            newUser->user_status=status;
            if(autoFile=="yes") {
                newUser->autoFileGet=true;
            } else if(autoFile=="no") {
                newUser->autoFileGet=false;
            }
            logs->append("Новый пользователь: "+newUser-
>get_user_name()+" со статусом "+newUser->user_status+" с AFG
"+autoFile);
            QImage img=QImage::fromData(icon_bits);
            testButton-
>setIcon(QIcon(QPixmap::fromImage(img)));
        }
    }
    messageLength = 0;
} else {
    logs->append("Пользователь "+usersInfo.value(socket)->
get_user_name()+" сменил имя на "+name+" А статус с
"+usersInfo.value(socket)->user_status+" на "+status+" А AFG на
"+autoFile);
    usersInfo.value(socket)->set_username(name);
    //TODO ТУТ ПРИХОДИТ ТОЛЬКО ТЕГ [USERNAME]: либо отправляем
остальные теги, либо обрабатываем здесь их по отдельности
    //           usersInfo.value(socket)->status=status;
    //           if(autoFile=="yes") {
    //               usersInfo.value(socket)->autoFileGet=true;
    //           } else if(autoFile=="no") {
    //               usersInfo.value(socket)->autoFileGet=false;
    //           }
    messageLength=0;
    return;
}
/*else if(message.contains("[ICON]")) {
    int needPos=message.indexOf("]");
    //ICON

    //аналог sendToAll() но со списком пользователей, чтобы
выгрузить оттуда статус, иконку и имя
} */ /*else if(message.contains("[STATUS]")) {
    int needPos=message.indexOf("]");
    QString status=message.mid(needPos+2, message.count()-1);
    User *user=usersInfo.value(socket);
    user->status=status;
}

```

```

        logs->append("Статус "+user->get_username()+" изменен на:
"+user->status);
        messageLength=0;
//        return;
    } */ else {
//        QString name=message.left(message.indexOf(":"));
        QString name=usersInfo.value(socket)->username;
//TODO при чтении xml подставлять вместо непонятных символов <
        QDomElement newLog=log(doc,
QDateTime::currentDateTime().toString(), socket-
>peerAddress().toString(), name, "simpleMessage",
message.mid(message.indexOf(">", 8)+1, message.count()));
        domElement.appendChild(newLog);

        message.prepend("[ "+socket->peerAddress().toString()+" ] ");
        if(mainData!=1)
            sendMessage(message);
        messageLength = 0;
    }
}

void MainWindow::logoutUser() {
//    sendToAll(tr("<em>Пользователь отключился.</em>"));
//which client wants to leave
QTcpSocket *socket = qobject_cast<QTcpSocket *>(sender());
if(socket == 0) {
    return; //i dunno m8, let's stop there
}

QString message="Пользователь отключился.";
QDomElement newLog=log(doc,
QDateTime::currentDateTime().toString(), socket-
>peerAddress().toString(), "SERVER", "serverMessage", message);
domElement.appendChild(newLog);

QByteArray packet;
QDataStream out(&packet, QIODevice::WriteOnly);
out.setVersion(QDataStream::Qt_4_1);

out<<(quint16)0<<(QString)"DISCONNECTEDUSER"<<usersInfo.value(socket)-
>username;
qDebug() <<"USERS"<<users;
qDebug() <<"USERSINFO"<<usersInfo;
for(std::size_t i=0; i<users.count(); ++i) {
    if(users[i]!=socket) {
//        users[i]->write(packet);
    }
}
users.removeOne(socket);
usersInfo.remove(socket);

```

```

        setWindowTitle(ip.toString() + ":" + QString::number(port) + "
" + QString::number(users.count()));
        sendMessage(tr("<em>Пользователь отключился.</em>"));

        socket->deleteLater(); //makes qt not lag; once the slot is done
executing, will remove socket
    }

void MainWindow::sendMessage(const QString &message) {
    QByteArray packet;
    QDataStream out(&packet, QIODevice::WriteOnly);

    out << (quint16) 0 << (QString) "SIMPLE_MESSAGE" << message;
    logs->append(message);
    for(std::size_t i=0; i<users.size(); ++i) {
        users[i]->write(packet);
        users[i]->flush();
    }
    //    out << (quint16) 0;
    //    out << message;
    //    qDebug() << "Sent to all " << message;
    //    out.device()->seek(0);
    //    out << (quint16) (packet.size() - sizeof(quint16));
    //    logs->append(message);
    //    for(int i = 0; i < users.size(); ++i) {
    //        users[i]->write(packet);
    //    }
}

QDomElement MainWindow::makeElement(QDomDocument *domDoc, const
QString &strName, const QString &strAttr=nullptr, const QString
&Text=nullptr) {
    QDomElement domElement=domDoc->createElement(strName);
    if(!strAttr.isEmpty()) {
        QDomAttr domAttr=domDoc->createAttribute("number");
        domAttr.setValue(strAttr);
        domElement.setAttributeNode(domAttr);
    }
    if(!Text.isEmpty()) {
        QDomText domText=domDoc->createTextNode(Text);
        domElement.appendChild(domText);
    }
    return domElement;
}

QDomElement MainWindow::log(QDomDocument *domDoc, const QString
&dateTime, const QString &ip, const QString &name, const QString
&type, const QString &message) {
    static int n=1;

```

```

QDomElement domElement=makeElement(domDoc, "log",
QString() .setNum(n));
domElement.appendChild(makeElement(domDoc, "name", "", name));
domElement.appendChild(makeElement(domDoc, "ip", "", ip));
domElement.appendChild(makeElement(domDoc, "dateAndTime",
"",dateTime));
    if(type=="simpleMessage") {
        domElement.appendChild(makeElement(domDoc, "message", "", message));
    } else if(type=="formattedMessage") {
        domElement.appendChild(makeElement(domDoc, "formattedMessage",
"", message));
    } else if(type=="imageMessage") {
        domElement.appendChild(makeElement(domDoc, "imageMessage", "", message));
    } else if(type=="fileMessage") {
        domElement.appendChild(makeElement(domDoc, "fileMessage", "", message));
    } else if(type=="serverMessage") {
        domElement.appendChild(makeElement(domDoc, "serverMessage",
"", message));
    }
    ++n;
    return domElement;
}

void MainWindow::saveXMLFile() {
    QFile file("logs.xml");
    if(file.open(QIODevice::WriteOnly)) {
        QTextStream(&file)<<doc->toString();
        file.close();
    }
}

```

Листинг-11 lab6_server.pro

```

QT      += core gui
QT+=widgets network sql xml
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000      # disables all
the APIs deprecated before Qt 6.0.0

SOURCES += \
    Server_Settings.cpp \
    main.cpp \
    mainwindow.cpp \
    user.cpp

```

```

HEADERS += \
    Server_Settings.h \
    mainwindow.h \
    user.h

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

```

Приложение В

Листинг-12 Dialog_About_Me.h

```

#ifndef INFODIALOG_H
#define INFODIALOG_H

#include <QDialog>
#include <QString>
#include <QLineEdit>
#include <QPushButton>
#include <QCheckBox>
#include <QLabel>
#include <QLayout>
#include <QMessageBox>

#include "QMenuBar"
#include "QMenu"
#include "QDialog"
#include "QPushButton"
#include "QLineEdit"
#include "QBoxLayout"
#include "QLabel"
#include "QListWidget"
class DialogAboutMe : public QDialog {

    Q_OBJECT

public:
    DialogAboutMe( QWidget* parent = 0 );

```

```

~DialogAboutMe();

private:
    QBoxLayout* layout;
    QLabel *imageLabel;
    QListWidget *lw;
    QPushButton *accept;
};

#endif // INFODIALOG_H

```

Листинг-13 Dialog_About_Me.cpp

```

#include "QMenuBar"
#include "QMenu"
#include "QDialog"
#include "QPushButton"
#include "QLineEdit"
#include "QBoxLayout"
#include "QLabel"
#include "QFrame"
#include "QListWidget"
#include "QString"
#include <QGridLayout>
DialogAboutMe::DialogAboutMe( QWidget* parent ) : QDialog(parent) {
    setWindowTitle("Info");

    layout = new QHBoxLayout;
    QGridLayout *l=new QGridLayout;

    imageLabel=new QLabel;
    QPixmap myPixmap=QPixmap("/home/user/Documents/I.jpg");
    imageLabel->setPixmap( myPixmap );
    layout->addWidget(imageLabel);

    lw = new QListWidget(this);

    lw->addItem("Author:");
    lw->addItem("Ushakov Aleksandr Nikolaevich SCS201");
    lw->addItem("Build date:");
    lw->addItem(__DATE__);
    lw->addItem(__TIME__);
    lw->addItem("The version of Qt that I was going to use:");
    lw->addItem(QT_VERSION_STR);
    lw->addItem("The version of Qt that is running:");
    lw->addItem(qVersion());

    accept=new QPushButton("OK", this);
    connect(accept, SIGNAL(clicked()), this, SLOT(accept()));
    layout->addWidget(lw);
    l->addLayout(layout,0,0);
}

```

```

l->addWidget(accept, 1, 0);
setLayout(l);

}

DialogAboutMe::~DialogAboutMe () {
}

```

Листинг-14 User_Info.h

```

#ifndef USERINFODIALOG_H
#define USERINFODIALOG_H

#include <QWidget>
#include <QDialog>
#include <QString>
#include <QLineEdit>
#include <QPushButton>
#include <QLabel>
#include <QLayout>
#include <QMessageBox>
#include <QtNetwork/QTcpServer>
#include <QMainWindow>

class User_Info : public QDialog {
    Q_OBJECT

public:
    User_Info(QWidget *parent=nullptr, QString name="", QString ip="",
              QString dateDateTime="", QString status="");
    ~User_Info();

public slots:
    void on_returnButton_clicked();

private:
    void initializeLayout();
    QPushButton *returnButton;
    QLabel *ipLabel;
    QLabel *dateTimeLabel;
    QLabel *statusLabel;
    QVBoxLayout *verticalLayout;
};

#endif // USERINFODIALOG_H

```

Листинг-15 User_Info.cpp

```
#include "userInfoDialog.h"
```

```

#include <QHBoxLayout>

UserInfoDialog::UserInfoDialog(QWidget *parent, QString name, QString ip,
QString dateTIme, QString status) : QDialog(parent) {
    // initializeWidgets(ip, dateTIme, status);
    ipLabel=new QLabel(tr("IP: ")+ip);
    dateTImeLabel=new QLabel(tr("Connection time and date: ")+dateTIme);
    statusLabel=new QLabel(tr("Status: ")+status);
    returnButton=new QPushButton(tr("&Accept"));
    setModal(true);
    QGridLayout *t=new QGridLayout(this);
    t->addWidget(ipLabel);
    t->addWidget(dateTImeLabel);
    t->addWidget(statusLabel);
    t->addWidget(returnButton);
    setLayout(t);
    setWindowTitle(name);
    connect(returnButton, SIGNAL(clicked()), this,
SLOT(on_returnButton_clicked()));
}

UserInfoDialog::~UserInfoDialog()=default;

//void UserInfoDialog::initializeWidgets(QString ip, QString dateTIme,
QString status) {

//}

void UserInfoDialog::initializeLayout() {
    verticalLayout=new QVBoxLayout();
    verticalLayout->addWidget(ipLabel);
    verticalLayout->addWidget(dateTImeLabel);
    verticalLayout->addWidget(statusLabel);
    verticalLayout->addWidget(returnButton);
    setLayout(verticalLayout);
}

void UserInfoDialog::on_returnButton_clicked() {
    hide();
    close();
}

```

Листинг-16 Full_Size.h

```

#ifndef IMAGEFULLDIALOG_H
#define IMAGEFULLDIALOG_H

#include <QDialog>
#include <QString>
#include <QLineEdit>
#include <QPushButton>

```

```

#include <QCheckBox>
#include <QLabel>
#include <QLayout>
#include <QMessageBox>

class Full_Size : public QDialog {
    Q_OBJECT

public:
    Full_Size(QWidget *parent=nullptr, QPixmap *image=nullptr);
    ~Full_Size();
    QLabel *imageFullLabel;
    QVBoxLayout *verticalLayout;

private:
    QPixmap *image;
};

#endif // IMAGEFULLDIALOG_H

```

Листинг-17 Full_Size.cpp

```

#include "Full_Size.h"

ImageFullDialog::ImageFullDialog(QWidget *parent, QPixmap *image_new)
    : QDialog(parent) {
    setModal(false);

    imageFullLabel=new QLabel(this);

    verticalLayout=new QVBoxLayout();
    verticalLayout->addWidget(imageFullLabel);

    setLayout(verticalLayout);

    setWindowTitle(tr("Full size"));
    image=image_new;

}

ImageFullDialog::~ImageFullDialog()=default;

```

Листинг-18 Color_Message.h

```

#ifndef MESSAGESCOLOR_H
#define MESSAGESCOLOR_H

#include <QWidget>
#include <QDialog>

```

```

#include <QString>
#include <QLineEdit>
#include <QPushButton>
#include <QLabel>
#include <QLayout>
#include <QMessageBox>
#include <QtNetwork/QTcpServer>
#include <QMainWindow>
#include <QColorDialog>

class Color_Message : public QDialog {
    Q_OBJECT

public:
    Color_Message(QWidget *parent=nullptr);
    ~Color_Message();
    QPushButton *messageColorButton;
    QPushButton *infoUserColorButton;
    QPushButton *acceptButton;
    QColor messageColor=Qt::black;
    QColor infoUsersColor=Qt::black;

public slots:
    void on_messageColorButton_clicked();
    void on_infoUserColorButton_clicked();

};

#endif // MESSAGESCOLOR_H

```

Листинг-19 Color_Message.cpp

```

#include "Color_Message.h"
#include <QHBoxLayout>

MessagesDialog::MessagesDialog(QWidget *parent) : QDialog(parent) {
    // initializeWidgets();
    acceptButton=new QPushButton(tr("&Accept"));
    messageColorButton=new QPushButton(tr("Choose color of message"));
    infoUserColorButton=new QPushButton(tr("Choose the color of the user
information in the message"));
    setModal(true);

    //initializeLayout();
    QGridLayout *t=new QGridLayout(this);
    t->addWidget(messageColorButton);
    t->addWidget(infoUserColorButton);
    t->addWidget(acceptButton);
    setLayout(t);
    setWindowTitle(tr("Color message"));

```

```

        connect(acceptButton, SIGNAL(clicked()), parent,
SLOT(on_acceptMessagesColor()));
        connect(messageColorButton, SIGNAL(clicked()), this,
SLOT(on_messageColorButton_clicked()));
        connect(infoUserColorButton, SIGNAL(clicked()), this,
SLOT(on_infoUserColorButton_clicked()));
}

MessagesDialog::~MessagesDialog()=default;

void MessagesDialog::on_messageColorButton_clicked() {
    QColor newColor=QColorDialog::getColor(messageColor);
    if(newColor.isValid()) {
        messageColor=newColor;
    }
}

void MessagesDialog::on_infoUserColorButton_clicked() {
    QColor newColor=QColorDialog::getColor(infoUsersColor);
    if(newColor.isValid()) {
        infoUsersColor=newColor;
    }
}

```

Листинг-20 Server_Settings.h

```

#ifndef SERVERDIALOG_H
#define SERVERDIALOG_H

#include <QWidget>
#include <QDialog>
#include <QString>
#include <QLineEdit>
#include <QPushButton>
#include <QLabel>
#include <QLayout>
#include <QMessageBox>
#include <QtNetwork/QTcpServer>
#include <QMainWindow>

class Server_Settings : public QDialog {
    Q_OBJECT

public:
    Server_Settings(QWidget *parent=nullptr);
    ~Server_Settings();
    QPushButton *acceptButton;
    QLineEdit *ipLineEdit;
    QLineEdit *portLineEdit;

```

```

private:
    QLabel *ipLabel;
    QLabel *portLabel;
};

#endif // SERVERDIALOG_H

```

Листинг-21 Server_Settings.cpp

```

#include "Server_Settings.h"
#include <QHBoxLayout>

ServerDialog::ServerDialog(QWidget *parent) : QDialog(parent) {

    ipLabel=new QLabel(tr("IP: "));
    portLabel=new QLabel(tr("Port: "));
    ipLineEdit=new QLineEdit();

    portLineEdit=new QLineEdit();

    acceptButton=new QPushButton(tr("&Accept"));

    setFocusProxy(ipLineEdit);

    QGridLayout *t=new QGridLayout(this);
    t->addWidget(ipLabel);
    t->addWidget(ipLineEdit);
    t->addWidget(portLabel);
    t->addWidget(portLineEdit);
    t->addWidget(acceptButton);
    setLayout(t);
    setWindowTitle(tr("Server Settings"));
    connect(acceptButton, SIGNAL(clicked()), parent, SLOT(on_accept()));
}

ServerDialog::~ServerDialog()=default;

```

Листинг-22 Send_User.h

```

#include <QWidget>
#include <QDialog>
#include <QString>
#include <QLineEdit>
#include <QPushButton>
#include <QLabel>

```

```

#include <QLayout>
#include <QMessageBox>
#include <QtNetwork/QTcpServer>
#include <QMainWindow>
#include <QProgressBar>
#include <QFile>
#include <QFileDialog>

class Send_User : public QDialog {
    Q_OBJECT

public:
    Send_User(QWidget *parent=nullptr, int number=0);
    ~Send_User();
    QByteArray file_bity;
    QString filename;
    QPushButton *selectFileButton;
    QPushButton *sendButton;
    QPushButton *acceptButton;
    bool fileSavingReady=false;
    QPushButton* noButton;
    QProgressBar* progressBar;
    QLabel* fileLabel;
    int numbeR;
    int bytes;
    QByteArray packet;
    QString filename_new;
    int clicked=0;

public:
    void on_selectFile();
    // void on_acceptButton_clicked();
    // void on_noButton_clicked();

private:
    // void initializeWidgets();
    // void initializeLayout();

    QVBoxLayout *verticalLayout;
};


```

Листинг-23 Send_User.cpp

```

#include "Send_User.h"
#include <QHBoxLayout>

FileSendDialog::FileSendDialog(QWidget *parent, int number) : QDialog(parent)
{

```

```

initializeWidgets();
setModal(false);

initializeLayout();
progressBar->setMinimum(0);
progressBar->setMaximum(100);
numberR=number;
setWindowTitle(tr("Send file"));
connect(sendButton, SIGNAL(clicked()), parent, SLOT(on_sendFile()));
connect(selectFileButton, SIGNAL(clicked()), this,
SLOT(on_selectFile()));

connect(acceptButton, SIGNAL(clicked()), this,
SLOT(on_acceptButton_clicked()));
connect(noButton, SIGNAL(clicked()), this, SLOT(on_noButton_clicked()));
acceptButton->hide();
noButton->hide();

}

void FileSendDialog::on_acceptButton_clicked() {
    clicked=1;
    progressBar->show();
    QString dir=QFileDialog::getExistingDirectory(this, tr("Choose file"),
"/home", QFileDialog::ShowDirsOnly | QFileDialog::DontResolveSymlinks);
    if(!dir.isNull()) {
        progressBar->setValue(10);
        if(QFileInfo(dir+"/"+filename_new+".txt").exists()) {
            int i=1;
            progressBar->setValue(15);
            bool numberFound=false;
            while(numberFound==false) {
                QString filename_copy=filename_new;
                filename_copy.append(QString::number(i)+".txt");
                if(!QFileInfo(dir+"/"+filename_copy).exists()) {
                    numberFound=true;
                    filename_new=filename_copy;
                    break;
                }
                ++i;
            }
            progressBar->setValue(20);

            QFile file(dir+"/"+filename_new);
            if(file.open(QIODevice::ReadWrite)) {
                progressBar->setValue(25);
                QTextStream stream(&file);
                stream<<packet;
                file.close();
            }
        }
    }
}

```

```

        progressBar->setValue(100);
        fileSavingReady=true;
    } else {
        progressBar->setValue(20);
        filename_new.append(".txt");
        QFile file(dir+"/"+filename_new);
        if(file.open(QIODevice::ReadWrite)) {
            progressBar->setValue(1);
            QTextStream stream(&file);
            stream<<packet;
            file.close();

            progressBar->setValue(100);
            fileSavingReady=true;
        }
    }
}

void FileSendDialog::on_noButton_clicked() {
    clicked=2;
    this->close();
}

FileSendDialog::~FileSendDialog() {
    delete fileLabel;
    delete sendButton;
    delete selectFileButton;
    delete verticalLayout;
}

void FileSendDialog::initializeWidgets() {
    fileLabel=new QLabel(tr("Empry file"));
    acceptButton= new QPushButton(tr("Accept"));
    noButton=new QPushButton(tr("Deny"));
    selectFileButton=new QPushButton(tr("Choose file"));
    sendButton=new QPushButton(tr("Send file"));
    progressBar=new QProgressBar(this);
}

void FileSendDialog::initializeLayout() {
    verticalLayout=new QVBoxLayout();
    verticalLayout->addWidget(fileLabel);
    verticalLayout->addWidget(selectFileButton);
    verticalLayout->addWidget(acceptButton);
    verticalLayout->addWidget(noButton);
    verticalLayout->addWidget(sendButton);
    verticalLayout->addWidget(progressBar);
    setLayout(verticalLayout);
}

```

```

void FileSendDialog::on_selectFile() {
    QString path=QFileDialog::getOpenFileName(this, tr("Open File"),
"/home/user/Desktop", tr("Text Files (*.txt)"));
    QFile file(path);
    QFileinfo fileInfo(file.fileName());
    filename=fileInfo.baseName();
    bytes=fileInfo.size();
    fileLabel->setText("Выбранный файл: "+filename);
    fileLabel->setText(filename);
    progressBar->setValue(10);

    QByteArray block;
    file.open(QIODevice::ReadOnly);
    QDataStream out(&block, QIODevice::WriteOnly);
    progressBar->setValue(20);
    out.setVersion(QDataStream::Qt_4_1);
    out<<file.readAll();
    progressBar->setValue(25);
    file_bity=block;
    progressBar->setValue(1);
    file.close();

}

```

Листинг-24 user.h

```

#ifndef USER_H
#define USER_H

#include <QByteArray>
#include <QVector>
#include <QString>
#include <QtNetwork/QTcpSocket>
#include <QIcon>
#include <QDateTime>

class User {
public:
    User();
    QString username="User";
    QTcpSocket user_socket;
    QString user_status="Availble";
    QImage user_icon;
    QString ip="127.0.0.1";
    QByteArray user_icon_64;
    bool autoFileGet=0;
    QDateTime connected_time;

private:
    //    QString username="Test";
    QByteArray user_id;

```

```

QVector<QByteArray> current_users;

public:
    void add_user(QByteArray);
    void remove_user(QByteArray);
    bool find_user(QByteArray);
    void clear_users();
    void set_user_id(QByteArray);
    void set_username(QString);
    void set_user_icon(QImage);
    QByteArray get_user_id();
    QString get_user_name();
};

#endif // USER_H

```

Листинг-25 user.cpp

```

#include "user.h"

User::User()=default;

void User::add_user(QByteArray contact) {
    current_users.push_back(contact);
}

void User::remove_user(QByteArray contact) {
    for(std::size_t i=0; i<current_users.size(); ++i) {
        if(current_users[i]==contact) {
            current_users.remove(i);
        }
    }
}

bool User::find_user(QByteArray contact) {
    for(std::size_t i=0; i<current_users.size(); ++i) {
        if(current_users[i]==contact) {
            return true;
        }
    }
    return false;
}

void User::clear_users() {
    current_users.clear();
}

void User::set_user_id(QByteArray user_id) {
    this->user_id=user_id;
}

void User::set_username(QString name) {
    this->username=name;
}

```

```

}

void User::set_user_icon(QImage image) {
    this->user_icon=image;
}

QByteArray User::get_user_id() {
    return user_id;
}

QString User::get_user_name() {
    return username;
}

```

Листинг-26 Status_User.h

```

#include <QWidget>
#include <QDialog>
#include <QString>
#include <QLineEdit>
#include <QPushButton>
#include <QLabel>
#include <QLayout>
#include <QMessageBox>
#include <QtNetwork/QTcpServer>
#include <QMainWindow>

class Status_User : public QDialog {

    Q_OBJECT

public:
    Status_User(QWidget *parent=nullptr);
    ~Status_User();
    QPushButton *acceptButton;
    QLineEdit *customStatusLineEdit;

private:
    QLabel *customStatusLabel;

};

```

Листинг-27 Status_User.cpp

```

#include "Status_User.h"
#include <QHBoxLayout>

CustomStatusDialog::CustomStatusDialog(QWidget *parent) : QDialog(parent) {

```

```

customStatusLabel=new QLabel(tr("Status: "));
customStatusLineEdit=new QLineEdit();
acceptButton=new QPushButton(tr("&Accept"));
setFocusProxy(customStatusLineEdit);
QGridLayout *t=new QGridLayout(this);
t->addWidget(customStatusLabel);
t->addWidget(customStatusLineEdit);
t->addWidget(acceptButton);
setLayout(t);

setWindowTitle(tr("Status setting"));
connect(acceptButton, SIGNAL(clicked()), parent,
SLOT(on_statusCustom()));
}

CustomStatusDialog::~CustomStatusDialog()=default;

```

Листинг-28 Dialog_About_Me.h

```

#include <QDialog>
#include <QString>
#include <QLineEdit>
#include <QPushButton>
#include <QCheckBox>
#include <QLabel>
#include <QLayout>
#include <QMessageBox>

#include "QMenuBar"
#include "QMenu"
#include "QDialog"
#include "QPushButton"
#include "QLineEdit"
#include "QBoxLayout"
#include "QLabel"
#include "QListWidget"
class DialogAboutMe : public QDialog {

    Q_OBJECT

public:
    DialogAboutMe( QWidget* parent = 0 );
    ~DialogAboutMe();

private:
    QBoxLayout* layout;
    QLabel *imageLabel;
    QListWidget *lw;
    QPushButton *accept;
};


```

Листинг-29 Dialog_About_Me.cpp

```
#include "QMenuBar"
#include "QMenu"
#include "QDialog"
#include "QPushButton"
#include "QLineEdit"
#include "QBoxLayout"
#include "QLabel"
#include "QFrame"
#include "QListWidget"
#include "QString"
#include <QGridLayout>
DialogAboutMe::DialogAboutMe( QWidget* parent ) : QDialog(parent) {
    setWindowTitle("Info");

    layout = new QHBoxLayout;
    QGridLayout *l=new QGridLayout;

    imageLabel=new QLabel;
    QPixmap myPixmap=QPixmap("/home/user/Documents/I.jpg");
    imageLabel->setPixmap( myPixmap );
    layout->addWidget(imageLabel);

    lw = new QListWidget(this);

    lw->addItem("Author:");
    lw->addItem("Ushakov Aleksandr Nikolaevich SCS201");
    lw->addItem("Build date:");
    lw->addItem(__DATE__);
    lw->addItem(__TIME__);
    lw->addItem("The version of Qt that I was going to use:");
    lw->addItem(QT_VERSION_STR);
    lw->addItem("The version of Qt that is running:");
    lw->addItem(qVersion());

    accept=new QPushButton("OK", this);
    connect(accept, SIGNAL(clicked()), this, SLOT(accept()));
    layout->addWidget(lw);
    l->addLayout(layout,0,0);
    l->addWidget(accept,1, 0);
    setLayout(l);

}
DialogAboutMe::~DialogAboutMe() {
```

Листинг-30 user_name.h

```
#include <QWidget>
```

```

#include <QDialog>
#include <QString>
#include <QLineEdit>
#include <QPushButton>
#include <QLabel>
#include <QLayout>
#include <QMessageBox>
#include <QtNetwork/QTcpServer>
#include <QMainWindow>

class User_Name : public QDialog {
    Q_OBJECT

public:
    User_Name(QWidget *parent=nullptr);
    ~User_Name();
    QPushButton *acceptButton;
    QLineEdit *usernameLineEdit;

private:
    QLabel *usernameLabel;
    QHBoxLayout *usernameHorizontalLayout;
    QVBoxLayout *verticalLayout;
};


```

Листинг-31 user_name.cpp

```

#include "user_name.h"
#include <QHBoxLayout>

UsernameDialog::UsernameDialog(QWidget *parent) : QDialog(parent) {

    usernameLabel=new QLabel(tr("User name: "));
    usernameLineEdit=new QLineEdit();
    acceptButton=new QPushButton(tr("&Accept"));
    setFocusProxy(usernameLineEdit);
    usernameHorizontalLayout=new QHBoxLayout();
    verticalLayout=new QVBoxLayout();
    verticalLayout->addLayout(usernameHorizontalLayout);
    verticalLayout->addWidget(acceptButton);
    usernameHorizontalLayout->addWidget(usernameLabel);
    usernameHorizontalLayout->addWidget(usernameLineEdit);
    QGridLayout *t=new QGridLayout(this);
    t->addWidget(usernameLabel);
    t->addWidget(usernameLineEdit);
    t->addWidget(acceptButton);
    setLayout(t);
    setWindowTitle(tr("Settings of user"));
    connect(acceptButton, SIGNAL(clicked()), parent,
SLOT(on_acceptUsername()));
}


```

```
UsernameDialog::~UsernameDialog() =default;
```

Листинг-32 client.h

```
#include <QtGui>
#include <QtNetwork>
#include "user_info.h"
#include "user.h"
#include <QMainWindow>
#include "server_settings.h"
#include "send_user.h"
#include "full_size.h"
#include "Color_Message.h"
#include "status_user.h"
#include <QString>
#include "Dialog_About_Me.h"
#include <QColorDialog>
#include "status_user.h"
#include <QFileDialog>
#include "user_name.h"
#include <QtMultimedia/QMediaPlayer>
#include "customStatusDialog.h"
#include "fileSendDialog.h"
#include "imageFullDialog.h"
#include <QDomDocument>
#include <QFontDialog>

class Client : public QMainWindow, private Ui::Client {

    Q_OBJECT

public:
    Client();
    void setBackColor(QColor);
    QColor getBackColor();
    QString username="Test";
    QString status="Buzy";
    QString autoFileGet="no";

    QImage icon;
    QString icon_64;
    void toggleDateTimeFlag();
    bool getDateFlag();
    void toggleIpFlag();
    bool getIpFlag();
    void setMessageColor(QColor);
    QColor getMessageColor();

protected:
    void closeEvent(QCloseEvent *event);
```

```

private:
    void updateUsers();
    void remove(QLayout *layout);
    void updatedUser();
    QString size_to_convert(int sizeBytes);
    void count_percent_send();
    void count_percent_get();
    void readSettings();
    void saveXMLLogs();
    QDomElement log(QDomDocument *domDoc, const QString &dateTime, const QString
&ip, const QString &name, const QString &type, const QString &message);
    QDomElement makeElement(QDomDocument *domDoc, const QString &strName, const
QString &stdAttr, const QString &Text);

private slots:
    void on_sendPhoto_clicked();
    void on_connectButton_clicked();
    void on_sendButton_clicked();
    void on_message_returnPressed();
    void dataReceived();
    void connectUser();
    void on_exitAction_triggered();
    void logOut();
    void socketError(QAbstractSocket::SocketError error);
    void on_accept();
    void on_serverButton_clicked();
    void on_usernameButton_clicked();
    void on_acceptUsername();
    void on_infoButton_clicked();
    void on_dateTimeButton_clicked();
    void on_ipButton_clicked();
    void on_actionBackColor_triggered();
    void on_actionMessagesColor_triggered();
    void on_acceptMessagesColor();
    void on_statusBarButton_clicked();
    void preLogOut();
    void on_photoButton_clicked();
    void on_statusAvailable();
    void on_statusAfk();
    void on_statusNotDisturb();
    void on_statusCustom();
    void on_statusCustomOpen();
    void on_showContextMenuUsers(const QPoint&);
    void openInfoUser();
    void openFileSending();
    void on_sendFile();
    void on_showContextMenuMessagesListUsers(const QPoint&);
    void openFullSizeImage();
    void saveImage();
    void on_saveLogs();
    void on_colorChangeButton_clicked();
    void on_fontChangeButton_clicked();
    void on_formatSendButton_clicked();
    void on_autoFileGetButton_clicked();

```

```

private:
quint16 port=45678;
    Server_Settings *sDialog;
    User_Name *uDialog;
    DialogAboutMe *infoDialog;
    Status_User *statusDialog;
    Color_Message *mDialog;
    bool dateTImeFlag=false;
    bool ipFlag=false;
    QColor color=Qt::white;
    QColor messageColor=Qt::black;
    QColor infoUsersColor=Qt::black;
    QList<User *> userList;
    int clientsOnline=0;
    bool firstClientTry=true;
    QMap<QByteArray, QString> iconsList;
    Send_User *fDialog;
    QLabel *amountGettingFiles;
    QLabel *fileGettingProgress;
    QLabel *fileGettingSize;
    QLabel *amountSendingFiles;
    QLabel *fileSendingProgress;
    QLabel *fileSendingSize;
    QHBoxLayout *gettingLayout;
    QHBoxLayout *sendingLayout;
    QVBoxLayout *statusBarLayout;
    int amountSendFiles=0;
    QAction *sAction;
    QAction *sendButton;
    QAction *sendPhoto;
    QAction *sendFormat;
    bool formatSending=false;
    QString id;
    QTcpSocket *socket;
    quint16 messageLength;
    User *user;
    QHostAddress ip=QHostAddress("127.0.0.1");
    int amountGetFiles=0;
    int amountSendedFiles=0;
    int amountGettedFiles=0;
    int percentProgressSend=0;
    int percentProgressGet=0;
    int sizeBytesSend=0;
    int sizeBytesGet=0;
    QList<Send_User*> fileSendList;
    int openedAmount=0;
    QDomDocument *doc;
    QDomElement domElement;
    QMap<int, QImage> photosList;
    int lastPos;
    ImageFullDialog *iDialog;
    QColor formattedColor=Qt::black;
    QFont formattedFont;

```

```
};
```

Листинг-33 client.cpp

```
#include "Client.h"

void Client::dataReceived() {
    int serverMes=0;
    QDataStream in(socket);

    if(messageLength ==0) {
        if(socket->bytesAvailable() < (int) sizeof(quint16)) {

            return;
        }

        in >> messageLength;
    }

    if(messageLength==0) {

        QString text;
        in>>text;

        if(text=="USERSONLINE") {

            userList.clear();
            QString names;
            QStringList namesS;
            QByteArray images_64;
            QList<QByteArray> imagesS_64;
            QByteArray image;
            in>>names;

            namesS=names.split(",");
            namesS.removeLast();
            namesS.removeOne(username);
            in>>images_64;
            imagesS_64=images_64.split(',');
            imagesS_64.removeLast();

            for(std::size_t i=0; i<namesS.count(); ++i) {
                User *newUser=new User();
                newUser->username=namesS[i];
                newUser-
>icon=QImage::fromData(QByteArray::fromBase64(imagesS_64[i]), "PNG");
                iconsList.insert(imagesS_64[i],namesS[i]);
                userList<<newUser;
            }
        }
    }
}
```

```

        updateUserList();
        messageLength = 0;
        return;
        serverMes=1;
    } else if(text=="SIMPLE_MESSAGE") {
        QString message;
        in>>message;
        // qDebug() << message;
        if(message=="<em>The user is disconnected!.</em>") {
            QString messageNew="The user is disconnected!";
            QDomElement newLog=log(doc,
QDateTime::currentDateTime().toString(), socket-
>peerAddress().toString(), "SERVER", "serverMessage", messageNew);
            domElement.appendChild(newLog);
            messagesList->append(message);
            updatedUser();
            messageLength=0;
            return;
        } else if(message=="<em>The user is connected!</em>") {
            QString messageNew="The user is connected!";
            QDomElement newLog=log(doc,
QDateTime::currentDateTime().toString(), socket-
>peerAddress().toString(), "SERVER", "serverMessage", messageNew);
            domElement.appendChild(newLog);
            messagesList->append(message);
            messageLength=0;
            return;
        }
        int needPos=message.indexOf(">");
        int needPos2=message.indexOf("<", needPos);
        int needPos3=message.indexOf("[");
        int needPos4=message.indexOf("]");
        int needPos5=message.indexOf(">", needPos2);
        QString name=message.mid(needPos+1, needPos2-needPos-1);
        QString ipString=message.mid(needPos3+1,needPos4-1);

        QDomElement newLog=log(doc,
QDateTime::currentDateTime().toString(), ipString, name,
"simpleMessage", message.mid(needPos5+1, message.count()));
        domElement.appendChild(newLog);

        name.prepend("<strong><font
color="+infoUsersColor.name()+">");
        name.append("</font></strong>");
        ipString.prepend("<font
color="+infoUsersColor.name()+">[");
        ipString.append("]</font>");
        QString messageString=message.mid(needPos5+1,
message.count());
        messageString.prepend("<font
color="+messageColor.name()+">");


```

```

messageString.append("</font>");
QString dateTimeString="";
QString finalMessage="";
finalMessage.append(name+": ");
finalMessage.append(messageString);
if(ipFlag) {
    finalMessage.prepend(ipString);
}
if(!ipFlag) {
    int firstIp=message.indexOf("[");
    int secondIp=message.indexOf("]");
    message.remove(firstIp, secondIp+1);
    message.prepend(" ");
}
if(dateTimeFlag) {

dateTimeString="\n"+QString::number(QDateTime::currentDateTime().date().day())+"."+QString::number(QDateTime::currentDateTime().date().month())+"."+QString::number(QDateTime::currentDateTime().date().year())+"."+QString::number(QDateTime::currentDateTime().time().hour())+":".+QString::number(QDateTime::currentDateTime().time().minute())+" ";
dateTimeString.prepend("<font
color="+infoUsersColor.name()+">");
dateTimeString.append("</font>");
finalMessage.prepend(dateTimeString);
}
message.prepend("<font color="+messageColor.name()+">");
message.append("</font>");
if(status!="Do not disturb") {
    QMediaPlayer *player=new QMediaPlayer;
    player-
>setMedia(QUrl::fromLocalFile("/home/user/Desktop/Ringtones/ring2.mp3"));
    player->setVolume(50);
    player->play();
}
messagesList->append(finalMessage);
messageLength = 0;
serverMes=1;
return;

} else if(text=="DISCONNECTEDUSER") {
QString usernameDisc;
in>>usernameDisc;
for(std::size_t i=0; i<userList.count(); ++i) {
    if(userList[i]->username==usernameDisc) {
        userList.removeAt(i);
    }
}
for(std::size_t i=0; i<usersList->count(); ++i) {
    if(usersList->item(i)->text()==usernameDisc) {

```

```

        usersList->removeItemWidget(usersList->item(i));
    }
}
} else if(text=="USERLIST_UPDATED") {

    userList.clear();
    QString all_names;
    QByteArray all_bytes;
    QStringList ips;
    QStringList times;
    QStringList statuses;
    in>>all_names;
    QByteArrayList bytes;
    in>>bytes;
    in>>ips;
    in>>times;
    in>>statuses;

    QStringList names=all_names.split(";");
    names.removeLast();
    User *deletedLaterUser;
    for(int i=0; i<names.count(); ++i) {
        User *newUser=new User();
        newUser->username=names[i];
        newUser-
>icon=QImage::fromData(QByteArray::fromBase64(bytes[i]), "PNG");
        newUser->ip=ips[i];
        newUser->connected_time=times[i];
        newUser->status=statuses[i];

        if(username==names[i])
            deletedLaterUser=newUser;
        userList<<newUser;
    }
    userList.removeOne(deletedLaterUser);
    updateUserList();
    messageLength = 0;

}

//МОЖНО УДАЛИТЬ ПО ИДЕЕ, DELETE
else if(text=="FILESENDED") {

    if(fDialog->isVisible())
        fDialog->progressBar->setValue(100);
    // QMessageBox::information(this, tr("Успех!"), tr("Вы
успешно отправили файл!"));
    ++amountSendedFiles;
    countPercentSend();
    messageLength = 0;
    return;
}

```

```

} else if(text=="ASKFORFILEGET") {
    QString filename;
    QString username;
    in>>filename;
    in>>username;
    if(status!="Do not disturb") {
        QMediaPlayer *player=new QMediaPlayer;
        player-
>setMedia(QUrl::fromLocalFile("/home/user/Desktop/Ringtones/ring3.mp3"));
        player->setVolume(50);
        player->play();
    }

    if(autoFileGet=="yes") {
        QByteArray block;
        QDataStream out(&block, QIODevice::WriteOnly);
        out.setVersion(QDataStream::Qt_4_1);
        out<<(quint16)0<<(QString)"FILEGETACCEPT";
        socket->write(block);

    } else if(autoFileGet=="no") {
        QMessageBox::StandardButton msgBox;
        msgBox=QMessageBox::question(this, "The file has
arrived", "Do you really want to accept a file with the name
"+filename+ " from "+username+"?", QMessageBox::Yes|QMessageBox::No);
        if(msgBox==QMessageBox::Yes) {
            QByteArray block;
            QDataStream out(&block, QIODevice::WriteOnly);
            out.setVersion(QDataStream::Qt_4_1);
            out<<(quint16)0<<(QString)"FILEGETACCEPT";
            socket->write(block);

            FileSendDialog *fDialogNew=new
FileSendDialog(this, openedAmount);
            fileSendList.append(fDialogNew);
            fDialogNew->selectFileButton->hide();
            fDialogNew->sendButton->hide();
            fDialogNew->fileLabel->setText(filename);
            fDialogNew->show();
            fDialog=fDialogNew;
            fDialog->progressBar->setValue(100);
        } else {
            QByteArray block;
            QDataStream out(&block, QIODevice::WriteOnly);
            out.setVersion(QDataStream::Qt_4_1);
            out<<(quint16)0<<(QString)"FILEGETDECLINE";
            socket->write(block);
        }
    }
}

```



```

        }
    } else {
        filename.append(".txt");
        QFile file("/home/user/"+filename);
        if(file.open(QIODevice::ReadWrite)) {
            QTextStream stream(&file);
            stream<<fileBlock;
            file.close();

            ++amountGettedFiles;
            countPercentGet();
        }
    }

QString ipString;
for(std::size_t i=0; i<userList.count(); ++i) {
    if(userList[i]->username==usernameSend) {
        ipString=userList[i]->ip;
    }
}
QDomElement newLog=log(doc,
QDateTime::currentDateTime().toString(), ipString, usernameSend,
"fileMessage", filename+"."+md5);
domElement.appendChild(newLog);
messagesList->append("You get file"+usernameSend+". File
name: "+filename+". Size: "+QString::number(size)+"Bytes");
messageLength = 0;
return;
} else if(text=="SERVERGOTFILE") {
    //
    messageLength = 0;
    return;
} else if(text=="CLIENTASKEDFORFILE") {
    //
    messageLength = 0;
    return;
} else if(text=="CLIENTACCEPTFILE") {
    qDebug() << "CLIENTACCEPTFILE";
    fDialog->progressBar->setValue(100);
    ++amountSendedFiles;
    countPercentSend();

    messageLength = 0;
    return;
} else if(text=="CLIENTDECLINEFILE") {
    qDebug() << "CLIENTDECLINEFILE";
    messageLength = 0;
    return;
}
//Тоже можно удалить, DELETE

```

```

    else if(text=="FILEFORYOU") {
        ++amountGetFiles;
        amountGettingFiles->setText("Number of files received:
"+QString::number(amountGetFiles));
        countPercentGet();
        QString filename;
        QByteArray block;
        QString username;
        in>>filename;
        in>>block;
        in>>username;
        block.remove(0,4);
        int bsize=block.count();
        sizeBytesGet+=bsize;
        fileGettingSize->setText(sizeConvert(sizeBytesGet));
        QString md5=QString(QCryptographicHash::hash((block),
QCryptographicHash::Md5).toHex()));

        if(status!="Do not disturb") {
            QMediaPlayer *player=new QMediaPlayer;
            player-
>setMedia(QUrl::fromLocalFile("/home/user/Desktop/Ringtones/ring4.mp3"
));
            player->setVolume(50);
            player->play();
        }

        FileSendDialog *gDialog=new FileSendDialog(this);
        if(autoFileGet=="no") {
            qDebug()<<"OPEN INFO WINDOW";
            gDialog->show();
            gDialog->fileLabel->setText("Do you agree to accept
the file: "+filename+".txt."+" From "+username+". Size
"+QString::number(bsize));
            gDialog->acceptButton->show();
            gDialog->noButton->show();
            gDialog->selectFileDialog->hide();
            gDialog->sendButton->hide();
            gDialog->progressBar->hide();
            gDialog->packet=block;
            gDialog->filename_new=filename;

            if(gDialog->fileSavingReady) {
                gDialog->progressBar->setValue(60);
                QByteArray blockForUser;
                QDataStream out(&blockForUser,
QIODevice::WriteOnly);
                out.setVersion(QDataStream::Qt_4_1);
                out<<(quint16)0<<(QString)"FILESENDED";
                socket->write(blockForUser);
            }
        }
    }
}

```

```

        gDialog->progressBar->setValue(80);
        ++amountGettedFiles;
        countPercentGet();

        gDialog->progressBar->setValue(100);
        messagesList->append("You get file from
"+username+". File name: "+filename+". Size: "+bsize+"Bytes");
    }
    messageLength = 0;
    return;
}

if(QFileInfo("/home/user/"+filename+".txt").exists()) {
    int i=1;
    bool numberFound=false;
    while(numberFound==false) {
        QString filename_copy=filename;
        filename_copy.append(QString::number(i)+".txt");

    if(!QFileInfo("/home/user/"+filename_copy).exists()) {
        numberFound=true;
        filename=filename_copy;
        break;
    }
    ++i;
}
QFile file("/home/user/"+filename);
if(file.open(QIODevice::ReadWrite)) {
    QTextStream stream(&file);
    stream<<block;
    file.close();

    if(gDialog->clicked==1)
        gDialog->progressBar->setValue(60);

    QByteArray blockForUser;
    QDataStream out(&blockForUser,
QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_1);
    out<<(quint16) 0<<(QString) "FILESENDED";
    if(gDialog->clicked==1)
        gDialog->progressBar->setValue(99);
    ++amountGettedFiles;
    countPercentGet();
    if(autoFileGet=="yes") {
        messagesList->append("You get file from
"+username+". File name: "+filename+". Size: "+file.size()+"Bytes");
        messageLength = 0;
        return;
    }
}

```

```

    } else {
        filename.append(".txt");
        QFile file("/home/user/"+filename);
        if(file.open(QIODevice::ReadWrite)) {
            QTextStream stream(&file);
            stream<<block;
            file.close();

            QByteArray blockForUser;
            QDataStream out(&blockForUser,
QIODevice::WriteOnly);
            out.setVersion(QDataStream::Qt_4_1);
            out<<(quint16)0<<(QString)"FILESENDED";
            ++amountGettedFiles;
            countPercentGet();
            if(autoFileGet=="yes") {
                messagesList->append("You get file from
"+username+". File name: "+filename+. Size: "+file.size()+"Bytes");
                messageLength = 0;
                return;
            }
        }
    }

    QString ipString;
    for(std::size_t i=0; i<userList.count(); ++i) {
        if(userList[i]->username==username) {
            ipString=userList[i]->ip;
        }
    }
    QDomElement newLog=log(doc,
QDateTime::currentDateTime().toString(), ipString, username,
"fileMessage", filename+", "+md5);
    domElement.appendChild(newLog);

} else if(text=="FORMATTED_MESSAGE") {
    if(status!="Do not disturb") {
        QMediaPlayer *player=new QMediaPlayer;
        player-
>setMedia(QUrl::fromLocalFile("/home/user/Desktop/Ringtones/ring4.mp3"));
        player->setVolume(50);
        player->play();
    }
    QString username;
    QString colorStr;
    QString fontName;
    int fontWeight;
    QString ip;
    QString message;
    in>>username;

```

```

        in>>colorStr;
        in>>fontName;
        in>>fontWeight;
        in>>ip;
        in>>message;

        QString
infoFormat= (" "+colorStr+", "+fontName+", "+fontWeight+" )  ";
        QDomElement newLog=log(doc,
QDateTime::currentDateTime().toString(), ip, username,
"formattedMessage", infoFormat+message);
        domElement.appendChild(newLog);

        QString dateTimeString;
        username.prepend("<strong><font
color="+infoUsersColor.name()+">");
        username.append("</font></strong>: ");

dateTimeString="\n"+QString::number(QDateTime::currentDateTime().date()
).day()+"."+QString::number(QDateTime::currentDateTime().date().month()
)+"."+QString::number(QDateTime::currentDateTime().date().year())+""
"+QString::number(QDateTime::currentDateTime().time().hour()):"+QString::number(QDateTime::currentDateTime().time().minute())+" ";
        if(messageColor!=Qt::black) {
            message.prepend("<font
color="+messageColor.name()+">");
            message.append("</font>");
        } else {
            message.prepend("<font color="+colorStr+" font-
size="+fontWeight+" font-family="+fontName+">");
            message.append("</font>");
        }
        message.prepend(username);
        if(ipFlag) {
            ip.prepend("<font color="+infoUsersColor.name()+">[");
            ip.append("]</font> ");
            message.prepend(ip);
        }
        if(dateTimeFlag) {
            dateTimeString.prepend("<font
color="+infoUsersColor.name()+">");
            dateTimeString.append("</font>");
            message.prepend(dateTimeString);
        }
        messagesList->append(message);
        messageLength = 0;
        return;

    }

else if(text=="UPDATED_USER_ICON") {

```

```

QString name;
QByteArray icon_64;
in>>name;
in>>icon_64;

for(std::size_t i=0; i<userList.count(); ++i) {
    if(userList[i]->username==name) {
        userList[i]-
>icon=QImage::fromData(QByteArray::fromBase64(icon_64), "PNG");
    }
}
updateUserList();
messageLength = 0;
serverMes=1;
return;
} else if(text=="PHOTO") {

    QString author;
    QByteArray image_64;
    QString ipString;
    in>>author;
    in>>ipString;
    in>>image_64;
    qDebug()<<"SENDED PHOTO"<<image_64;
    QImage
image=QImage::fromData(QByteArray::fromBase64(image_64), "PNG");
    QImage image_new=image.scaled(320,240);
    QTextCursor newCursor=messagesList->textCursor();
    newCursor.movePosition(QTextCursor::End);

    photosList.insert(newCursor.position(), image);

    if(status!="Do not disturb") {
        QMediaPlayer *player=new QMediaPlayer;
        player-
>setMedia(QUrl::fromLocalFile("/home/user/Desktop/Ringtones/ring5.mp3"));
        player->setVolume(50);
        player->play();
    }
    messagesList->setTextCursor(newCursor);
    messagesList->textCursor().insertBlock();
    messagesList->textCursor().insertImage(image_new);

    QBuffer buffer;
    buffer.open(QIODevice::WriteOnly);
    image_new.save(&buffer, "PNG");
    QString encoded=buffer.data().toBase64();
}

```

```

        QDomElement newLog=log(doc,
QDateTime::currentDateTime().toString(), ipString, author,
"imageMessage", encoded);
        domElement.appendChild(newLog);

        messageLength = 0;
        return;
    }
}

void Client::connectUser() {
    messagesList->append(tr("<em>Successful connection</em>"));
    connectAction->setEnabled(true);

    QByteArray packet;
    QDataStream out(&packet, QIODevice::WriteOnly);

    QString content="";
    QString messageToSend = "[USERNAME] "+username
+", [STATUS]" +status+", [AFG]" +autoFileGet+", [ICON]" +content;
    out << (quint16) 0;
    out << messageToSend;
    out.device()->seek(0);
    out << (quint16) (packet.size() - sizeof(quint16));
    socket->write(packet);

    setWindowTitle(ip.toString() + ":" +QString::number(port) +"
"+status);

}

void Client::preLogOut() {
    messagesList->append(tr("<em>The user is disconnected.</em>"));
    setWindowTitle("Client");
    QByteArray block;
    QDataStream out(&block, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_1);
    out<<(quint16) 0<<(QString) "DISCONNECT"<<username;
    socket->write(block);
    userList.clear();
    usersList->clear();
    logOut();
}

void Client::logOut() {
    socket->disconnectFromHost();
}

void Client::socketError(QAbstractSocket::SocketError error) {
    switch(error) {

```

```

        case QAbstractSocket::HostNotFoundError:
            messagesList->append(tr("Error: server does not
exit.</em>"));
            break;
        case QAbstractSocket::ConnectionRefusedError:
            messagesList->append(tr("Error: connection is
destroyed.</em>"));
            break;
        case QAbstractSocket::RemoteHostClosedError:
            messagesList->append(tr("Error: connection is
closed.</em>"));
            break;
        default:
            messagesList->append(tr("Error: ") + socket-
>errorString() + tr("</em>"));
    }

    connectAction->setEnabled(true);
}

void Client::on_actionBackColor_triggered() {
    QColor newColor=QColorDialog::getColor(getBackColor());
    if(newColor.isValid()) {
        setBackColor(newColor);
    }
}

void Client::on_showContextMenuUsers(const QPoint &point) {
    QPoint globalPos=usersList->mapToGlobal(point);
    QMenu contextMenu;
    contextMenu.addAction("Info", this, SLOT(openInfoUser()));
    contextMenu.addAction("Send File", this, SLOT(openFileSending()));
    contextMenu.exec(globalPos);
}

void Client::on_showContextMenuMessagesListUsers(const QPoint &point)
{
    QPoint globalPos=messagesList->mapToGlobal(point);
    QMenu contextMenu;
    contextMenu.addAction("Open in full window", this,
SLOT(openFullSizeImage()));
    contextMenu.addAction("Save image", this, SLOT(saveImage()));
    contextMenu.exec(globalPos);
    lastPos=messagesList->textCursor().position();
}

void Client::openFullSizeImage() {
    QImage image;
    QBuffer buffer;
    ImageFullDialog *iDialog=new ImageFullDialog(this);

```

```

buffer.open(QIODevice::WriteOnly);
if(photosList.value(lastPos).isNull()) {
    if(photosList.value(lastPos-1).isNull()) {
        if(photosList.value(lastPos-2).isNull()) {
            if(photosList.value(lastPos-3).isNull()) {
                if(photosList.value(lastPos-4).isNull()) {
                    if(photosList.value(lastPos-5).isNull()) {
                        //nothing
                    } else {
                        photosList.value(lastPos-5).save(&buffer,
"PNG");
                    }
                } else {
                    photosList.value(lastPos-4).save(&buffer,
"PNG");
                }
            } else {
                photosList.value(lastPos-3).save(&buffer, "PNG");
                image=photosList.value(lastPos-3);
            }
        } else {
            photosList.value(lastPos-2).save(&buffer, "PNG");
            image=photosList.value(lastPos-2);
        }
    } else {
        photosList.value(lastPos-1).save(&buffer, "PNG");
        image=photosList.value(lastPos-1);
    }
} else {
    photosList.value(lastPos).save(&buffer, "PNG");
    image=photosList.value(lastPos);
}
QLabel *image_label=new QLabel(this);
image_label->setPixmap(QPixmap::fromImage(image));

iDialog->imageFullLabel->setPixmap(QPixmap::fromImage(image));
iDialog->imageFullLabel->setWindowFlags(Qt::Window);
iDialog->verticalLayout->addWidget(image_label);
iDialog->show();
QString encoded=buffer.data().toBase64();
// qDebug()<<encoded<<"YES!";
}

void Client::saveImage() {
    QImage image;
    QBuffer buffer;
    buffer.open(QIODevice::WriteOnly);
    if(photosList.value(lastPos).isNull()) {
        if(photosList.value(lastPos-1).isNull()) {

```

```

    if(photosList.value(lastPos-2).isNull()) {
        if(photosList.value(lastPos-3).isNull()) {
            if(photosList.value(lastPos-4).isNull()) {
                if(photosList.value(lastPos-5).isNull()) {
                    //nothing
                } else {
                    photosList.value(lastPos-5).save(&buffer,
                    "PNG");
                }
            } else {
                photosList.value(lastPos-4).save(&buffer,
                "PNG");
            }
        } else {
            photosList.value(lastPos-3).save(&buffer, "PNG");
            image=photosList.value(lastPos-3);
        }
    } else {
        photosList.value(lastPos-2).save(&buffer, "PNG");
        image=photosList.value(lastPos-2);
    }
} else {
    photosList.value(lastPos-1).save(&buffer, "PNG");
    image=photosList.value(lastPos-1);
}
} else {
    photosList.value(lastPos).save(&buffer, "PNG");
    image=photosList.value(lastPos);
}

QString fileName = QFileDialog::getSaveFileName(this, tr("Save
Image File"), QString(), tr("Images (*.png)"));
if(!fileName.isEmpty()) {
    image.save(fileName);

}
}

void Client::openInfoUser() {
    QString name=usersList->currentItem()->text();
    QString dateTime;
    QString ip;
    QString status;
    for(std::size_t i=0; i<userList.count(); ++i) {
        if(userList[i]->username==name) {
            dateTime=userList[i]->connected_time;
            ip=userList[i]->ip;
            status=userList[i]->status;
        }
    }
}

```

```

        }
        UserInfoDialog *userInfoDialog=new UserInfoDialog(this, name, ip,
dateTime, status);
        userInfoDialog->show();
    }

void Client::openFileSending() {
    FileSendDialog *fDialogNew=new FileSendDialog(this, openedAmount);
    fileSendList.append(fDialogNew);
    fDialogNew->show();
    fDialog=fDialogNew;
    ++openedAmount;
    ++amountSendFiles;
    amountSendingFiles->setText("Number of files to be sent:
"+QString::number(amountSendFiles));
    countPercentSend();
}

QString Client::sizeConvert(int bsize) {
    int bytes=bsize;
    float kbytes=bytes/1024;
    float mbytes=bytes/1024;
    QString result="Same size files: Bytes "+QString::number(bytes)+" KB
"+QString::number(kbytes)+" MB "+QString::number(mbytes);
    return result;
}

void Client::countPercentSend() {
    int percent=(100*amountSendedFiles)/amountSendFiles;
    percentProgressSend=percent;
    QString label="Progress of sending files:
"+QString::number(percent)+"%";
    fileSendingProgress->setText(label);
}

void Client::countPercentGet() {
    int percent=(100*amountGettedFiles)/amountGetFiles;
    percentProgressGet=percent;
    QString label="File receipt progress:
"+QString::number(percent)+"%";
    fileGettingProgress->setText(label);
}

void Client::on_sendFile() {
    sizeBytesSend+=fDialog->bytes;
    fileSendingSize->setText(sizeConvert(sizeBytesSend));
    fDialog->progressBar->setValue(35);
    QByteArray block=fDialog->file_bity;
    QString filename=fDialog->filename;
    quint16 size=fDialog->bytes;
    QString username=usersList->currentItem()->text();
}

```

```

fDialog->progressBar->setValue(50);
QByteArray packet;
QDataStream out(&packet, QIODevice::WriteOnly);
out.setVersion(QDataStream::Qt_4_1);

out<<(quint16) 0<<(QString) "FILEFORSERVER"<<username<<filename<<size<<block;
    fDialog->progressBar->setValue(75);
    socket->write(packet);
    fDialog->progressBar->setValue(90);
}

QDomElement Client::makeElement(QDomDocument *domDoc, const QString &strName, const QString &strAttr=QString::null, const QString &Text=QString::null) {
    QDomElement domElement=domDoc->createElement(strName);
    if(!strAttr.isEmpty()) {
        QDomAttr domAttr=domDoc->createAttribute("number");
        domAttr.setValue(strAttr);
        domElement.setAttributeNode(domAttr);
    }
    if(!Text.isEmpty()) {
        QDomText domText=domDoc->createTextNode(Text);
        domElement.appendChild(domText);
    }
    return domElement;
}

QDomElement Client::log(QDomDocument *domDoc, const QString &dateTime,
const QString &ip, const QString &name, const QString &type, const QString &message) {
    static int n=1;
    QDomElement domElement=makeElement(domDoc, "log",
QString().setNum(n));
    domElement.appendChild(makeElement(domDoc, "name", "", name));
    domElement.appendChild(makeElement(domDoc, "ip", "", ip));
    domElement.appendChild(makeElement(domDoc, "dateAndTime",
"", dateTime));
    if(type=="simpleMessage") {
        domElement.appendChild(makeElement(domDoc, "message", "", message));
    } else if(type=="formattedMessage") {
        domElement.appendChild(makeElement(domDoc, "formattedMessage",
"", message));
    } else if(type=="imageMessage") {
        domElement.appendChild(makeElement(domDoc, "imageMessage", "", message));
    } else if(type=="fileMessage") {
        domElement.appendChild(makeElement(domDoc, "fileMessage", "", message));
    }
}

```

```

        } else if(type=="serverMessage") {
            domElement.appendChild(makeElement(domDoc, "serverMessage",
"\"", message));
        }
        ++n;
        return domElement;
    }

void Client::saveXMLLogs() {
    QFile file("logs.xml");
    if(file.open(QIODevice::WriteOnly)) {
        QTextStream(&file)<<doc->toString();
        file.close();
    }
}

void Client::on_saveLogs() {
    saveXMLLogs();
}

void Client::on_autoFileGetButton_clicked() {
    if(fileAutoAction->isChecked()) {
        autoFileGet="yes";
    } else {
        autoFileGet="no";
    }
}

Client::Client() {
    setupUi(this);

    setWindowTitle("Client");
    QActionGroup *groupStatuses=new QActionGroup(this);
    groupStatuses->addAction(availableAction);
    groupStatuses->addAction(afkAction);
    groupStatuses->addAction(notDisturbAction);
    groupStatuses->addAction(customAction);
    groupStatuses->setExclusive(true);
    messagesList->setStyleSheet("background-color: \"");
    infoDialog=new DialogAboutMe(this);
    sDialog=new ServerDialog(this);
    uDialog=new UsernameDialog(this);
    mDialog=new MessagesDialog(this);
    statusDialog=new CustomStatusDialog(this);
    iDialog=new ImageFullDialog(this);
    sAction=serverAction;
    sAction->setText("Server:
"+ip.toString()+":"+QString::number(port));
    socket=new QTcpSocket(this);
    connect(messagesColorAction, SIGNAL(triggered()), this,
SLOT(on_actionMessagesColor_triggered()));
}

```

```

        connect(backColorAction, SIGNAL(triggered()), this,
SLOT(on_actionBackColor_triggered()));
        connect(ipAction, SIGNAL(triggered()), this,
SLOT(on_ipButton_clicked()));
        connect(dateTimeAction, SIGNAL(triggered()), this,
SLOT(on_dateTimeButton_clicked()));
        connect(infoAction, SIGNAL(triggered()), this,
SLOT(on_infoButton_clicked()));
        connect(usernameAction, SIGNAL(triggered()), this,
SLOT(on_usernameButton_clicked()));
        connect(connectAction, SIGNAL(triggered()), this,
SLOT(on_connectButton_clicked()));
        connect(sAction, SIGNAL(triggered()), this,
SLOT(on_serverButton_clicked()));
        connect(socket, SIGNAL(readyRead()), this, SLOT(dataReceived()));
        connect(socket, SIGNAL(connected()), this, SLOT(connectUser()));
        connect(disconnectAction, SIGNAL(triggered()), this,
SLOT(preLogOut()));
        connect(socket, SIGNAL(disconnected()), this, SLOT(logOut()));
        connect(socket, SIGNAL(error(QAbstractSocket::SocketError)), this,
SLOT(socketError(QAbstractSocket::SocketError)));
        connect(statusBarAction, SIGNAL(triggered()), this,
SLOT(on_statusBarButton_clicked()));
        connect(photoAction, SIGNAL(triggered()), this,
SLOT(on_photoButton_clicked()));
        connect(availableAction, SIGNAL(triggered()), this,
SLOT(on_statusAvailable()));
        connect(afkAction, SIGNAL(triggered()), this,
SLOT(on_statusAfk()));
        connect(notDisturbAction, SIGNAL(triggered()), this,
SLOT(on_statusNotDisturb()));
        connect(customAction, SIGNAL(triggered()), this,
SLOT(on_statusCustomOpen()));
        connect(saveLogsAction, SIGNAL(triggered()), this,
SLOT(on_saveLogs()));
        connect(exitAction, SIGNAL(triggered()), this,
SLOT(on_exitAction_triggered()));
        connect(messageColorButton, SIGNAL(clicked()), this,
SLOT(on_colorChangeButton_clicked()));
        connect(fontButton, SIGNAL(clicked()), this,
SLOT(on_fontChangeButton_clicked()));
        connect(fileAutoAction, SIGNAL(triggered()), this,
SLOT(on_autoFileGetButton_clicked()));

        messageLength = 0;
        sendButtonList->setPopupMode(QToolButton::MenuButtonPopup);
        sendButton=new QAction("Send message");
        sendPhoto=new QAction("Send photo");
        sendFormat=new QAction("Formatted message");
        sendButtonList->setDefaultAction(sendButton);
        sendButtonList->addAction(sendButton);
    
```

```

sendButtonList->addAction(sendPhoto);
sendButtonList->addAction(sendFormat);
connect(sendButton, SIGNAL(triggered()), this,
SLOT(on_sendButton_clicked()));
connect(sendPhoto, SIGNAL(triggered()), this,
SLOT(on_sendPhoto_clicked()));
connect(sendFormat, SIGNAL(triggered()), this,
SLOT(on_formatSendButton_clicked()));

usersList->setContextMenuPolicy(Qt::CustomContextMenu);
connect(usersList, SIGNAL(customContextMenuRequested(QPoint)),
this, SLOT(on_showContextMenuUsers(QPoint)));

messagesList->setContextMenuPolicy(Qt::CustomContextMenu);
connect(messagesList, SIGNAL(customContextMenuRequested(QPoint)),
this, SLOT(on_showContextMenuMessagesListUsers(QPoint)));

fontButton->hide();
messageColorButton->hide();

amountGettingFiles=new QLabel("Number of files received: 0");
fileGettingProgress=new QLabel("File receipt progress: 0%");
fileGettingSize=new QLabel("The size of the received files: 0 B");

amountSendingFiles=new QLabel("Number of files transferred: 0");
fileSendingProgress=new QLabel("Progress of sending files: 0%");
fileSendingSize=new QLabel("The size of the files being sent: 0
B");

statusBar()->addWidget(amountGettingFiles);
statusBar()->addWidget(fileGettingProgress);
statusBar()->addWidget(fileGettingSize);
statusBar()->addWidget(amountSendingFiles);
statusBar()->addWidget(fileSendingProgress);
statusBar()->addWidget(fileSendingSize);

doc=new QDomDocument("logs");
domElement=doc->createElement("logs");
doc->appendChild(domElement);

readSettings();

}

void Client::closeEvent(QCloseEvent *event) {
    event->ignore();
    on_exitAction_triggered();
}

void Client::readSettings() {

```

```

QSettings settings("/home/user/Desktop/ChatClient.config",
QSettings::IniFormat);

setBackColor(qvariant_cast<QColor>(settings.value("background_message_color")));
setMessageColor(qvariant_cast<QColor>(settings.value("message_color")));
username=qvariant_cast<QString>(settings.value("username"));
icon_64=qvariant_cast<QString>(settings.value("icon_64"));
status=qvariant_cast<QString>(settings.value("status"));

autoFileGet=qvariant_cast<QString>(settings.value("auto_get_file"));
ip=QHostAddress(qvariant_cast<QString>(settings.value("ip")));
port=qvariant_cast<quint16>(settings.value("port"));
ipFlag=qvariant_cast<bool>(settings.value("show_message_ip"));

dateTimeFlag=qvariant_cast<bool>(settings.value("show_message_time"));

    if(autoFileGet=="yes") {
        fileAutoAction->setChecked(true);
    }
    if(status=="Available") {
        availableAction->setChecked(true);
    } else if(status=="Away") {
        afkAction->setChecked(true);
    } else if(status=="Do not disturb") {
        notDisturbAction->setChecked(true);
    } else {
        if(status.length()>16) {
            QString name=status.left(16)+"...";
            customAction->setText(name);
        } else {
            customAction->setText(status);
        }
        customAction->setChecked(true);
    }

    if(ipFlag) {
        ipAction->setChecked(true);
    }
    if(dateTimeFlag) {
        dateTimeAction->setChecked(true);
    }
}

void Client::on_exitAction_triggered() {
    QSettings settings("/home/user/Desktop/ChatClient.config",
QSettings::IniFormat);
    settings.setValue("background_message_color", getBackColor());
}

```

```

        settings.setValue("message_color", getMessageColor());
        settings.setValue("auto_get_file", autoFileGet);
        settings.setValue("username", username);
        settings.setValue("icon_64", icon_64);
        settings.setValue("status", status);
        settings.setValue("ip", ip.toString());
        settings.setValue("port", port);
        settings.setValue("show_message_ip", ipFlag);
        settings.setValue("show_message_time", dateTImeFlag);

    QApplication::quit();
}

void Client::on_photoButton_clicked() {
    QString path=QFileDialog::getOpenFileName(this, tr("Open Image"),
"/home/user/Desktop/images", tr("Image Files (*.png)"));
    icon=QImage(path);

    updatedUser();
}

void Client::updatedUser() {
//формируем 64 от ИКОНКИ
    QBuffer buffer;
    QByteArray block;
    QDataStream out(&block, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_1);
    buffer.open(QIODevice::WriteOnly);
    icon.save(&buffer, "PNG");
    QString encoded=buffer.data().toBase64();
    icon_64=encoded;

out<<(quint16)0<<(QString)"USER_UPDATED"<<username<<encoded<<status;
    socket->write(block);
}

void Client::on_sendPhoto_clicked() {
    formatSending=false;
    fontButton->hide();
    messageColorButton->hide();
    sendButtonList->setDefaultAction(sendPhoto);

    QString path=QFileDialog::getOpenFileName(this, tr("Open Image"),
"/home/user/Desktop/images", tr("Image Files (*.png)"));
    QImage image(path);
    QBuffer buffer;
    QByteArray block;
    QDataStream out(&block, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_1);
    buffer.open(QIODevice::WriteOnly);
    image.save(&buffer, "PNG");
}

```

```

    QString encoded=buffer.data().toBase64();
    out<<(quint16)0<<(QString)"PHOTO"<<username<<encoded;
    socket->write(block);
}

void Client::on_statusAvailable() {
    status="Available";
    setWindowTitle(ip.toString()+":"+QString::number(port)+""
"+status);
    updatedUser();
}

void Client::on_statusAfk() {
    status="Away";
    setWindowTitle(ip.toString()+":"+QString::number(port)+""
"+status);
    updatedUser();
}

void Client::on_statusNotDisturb() {
    status="Do not disturb";
    setWindowTitle(ip.toString()+":"+QString::number(port)+""
"+status);
    updatedUser();
}

void Client::on_statusCustomOpen() {
    statusDialog->customStatusLineEdit->setText(status);
    statusDialog->show();
}

void Client::on_statusCustom() {
    status=statusDialog->customStatusLineEdit->text();
    if(status.length()>16) {
        customAction->setText(status.right(16)+"...");
    } else {
        customAction->setText(status);
    }
    setWindowTitle(ip.toString()+":"+QString::number(port)+""
"+status);
    updatedUser();
    statusDialog->hide();
}

void Client::on_statusBarButton_clicked() {
    statusBar()->setVisible(!statusBar()->isVisible());
}

void Client::setMessageColor(QColor newColor) {
    messageColor=newColor;
}

```

```

QColor Client::getMessageColor() {
    return messageColor;
}

void Client::on_actionMessagesColor_triggered() {
    mDialog->show();
}

void Client::on_acceptMessagesColor() {
    messageColor=mDialog->messageColor;
    infoUsersColor=mDialog->infoUsersColor;

    mDialog->hide();
}

void Client::setBackColor(QColor newColor) {
    messagesList->setStyleSheet("background-color: "+newColor.name());
    color=newColor;
}

QColor Client::getBackColor() {
    return color;
}

void Client::on_ipButton_clicked() {
    toggleIpFlag();
}

void Client::on_dateTimeButton_clicked() {
    toggleDateTimeFlag();
}

void Client::toggleIpFlag() {
    ipFlag=! (getIpFlag());
}

bool Client::getIpFlag() {
    return ipFlag;
}

void Client::toggleDateTimeFlag() {
    dateTimeFlag=! (getDateTimeFlag());
}

bool Client::getDateTimeFlag() {
    return dateTimeFlag;
}

void Client::on_infoButton_clicked() {
    infoDialog->show();
}

```

```

}

void Client::on_usernameButton_clicked() {
    uDialog->usernameLineEdit->setText(username);
    uDialog->show();
}

void Client::on_acceptUsername() {
    QString oldUsername=username;
    if (uDialog->usernameLineEdit->text().isEmpty()) {
        QMessageBox::information(uDialog, tr("Empty file"), tr("repeat
action"));
        return;
    }
    username=uDialog->usernameLineEdit->text();

    uDialog->hide();
    updatedUser();
}

void Client::on_serverButton_clicked() {
    sDialog->ipLineEdit->setText(ip.toString());
    sDialog->portLineEdit->setText(QString::number(port));
    sDialog->show();
}

void Client::on_accept() {
    if (sDialog->ipLineEdit->text().isEmpty() || sDialog->portLineEdit-
>text().isEmpty()) {
        QMessageBox::information(sDialog, tr("Empty field"),
tr("repeat action."));
        return;
    }
    ip=QHostAddress(sDialog->ipLineEdit->text());
    port=sDialog->portLineEdit->text().toUInt();

    sDialog->hide();
    serverAction->setText("Server: "+ip.toString()+" :
"+QString::number(port));
}

void Client::on_connectButton_clicked() {
    messagesList->append(tr("<em>Is connecting...</em>"));
    connectAction->setEnabled(false);

    socket->abort();
    socket->connectToHost(ip, port);

    QByteArray packet;
}

```

```

QBuffer buffer;
buffer.open(QIODevice::WriteOnly);
icon.save(&buffer, "PNG");
QString encoded=buffer.data().toBase64();
QDataStream out(&packet, QIODevice::WriteOnly);
out.setVersion(QDataStream::Qt_4_1);

out<<(quint16) 0<<(QString) "USERLIST_UPDATE"<<username<<encoded<<status
;
    socket->write(packet);
}

void Client::on_colorChangeButton_clicked() {
    QColor newColor=QColorDialog::getColor(formattedColor);
    if(newColor.isValid()) {
        formattedColor=newColor;
    }
}

void Client::on_fontChangeButton_clicked() {
    bool userChoseFont;
    formattedFont=QFontDialog::getFont(&userChoseFont, message-
>font(), this);
    if (!userChoseFont)
        return;
}

void Client::on_formatSendButton_clicked() {
    if(formatSending) {
        QByteArray packet;
        QDataStream out(&packet, QIODevice::WriteOnly);

        out<<(quint16) 0<<(QString) "FORMATTED_MESSAGE"<<username<<(QString) form-
        attedColor.name()<<formattedFont.family()<<formattedFont.weight()<<mes-
        sage->text();
        socket->write(packet);
        message->clear();
        message->setFocus();
    } else {
        sendButtonList->setDefaultAction(sendFormat);
        fontButton->show();
        messageColorButton->show();
        formatSending=true;
    }
}

void Client::on_sendButton_clicked() {
    formatSending=false;
    fontButton->hide();
    messageColorButton->hide();
    sendButtonList->setDefaultAction(sendButton);
}

```

```

QByteArray packet;
QDataStream out(&packet, QIODevice::WriteOnly);

QString messageToSend = "<strong>" + username + "</strong>" +
message->text();
out << (quint16) 0;
out << messageToSend;
out.device()->seek(0);
out << (quint16) (packet.size() - sizeof(quint16));

socket->write(packet);
message->clear();
message->setFocus();
}

void Client::on_message_returnPressed() {
    on_sendButton_clicked();
}

void Client::remove(QLayout *layout) {
    QLayoutItem *child;
    while(layout->count() != 0) {
        child=layout->takeAt(0);
        if(child->layout()!=0) {
            remove(child->layout());
        } else if(child->widget()!=0) {
            delete child->widget();
        }
        delete child;
    }
}

void Client::updateUserList() {
    usersList->clear();

    for(int i=0; i<userList.count(); ++i) {
        QListWidgetItem *item=new
        QListWidgetItem(QIcon(QPixmap::fromImage(userList[i]->icon)),
        userList[i]->username);
        usersList->addItem(item);
    }
}

```

