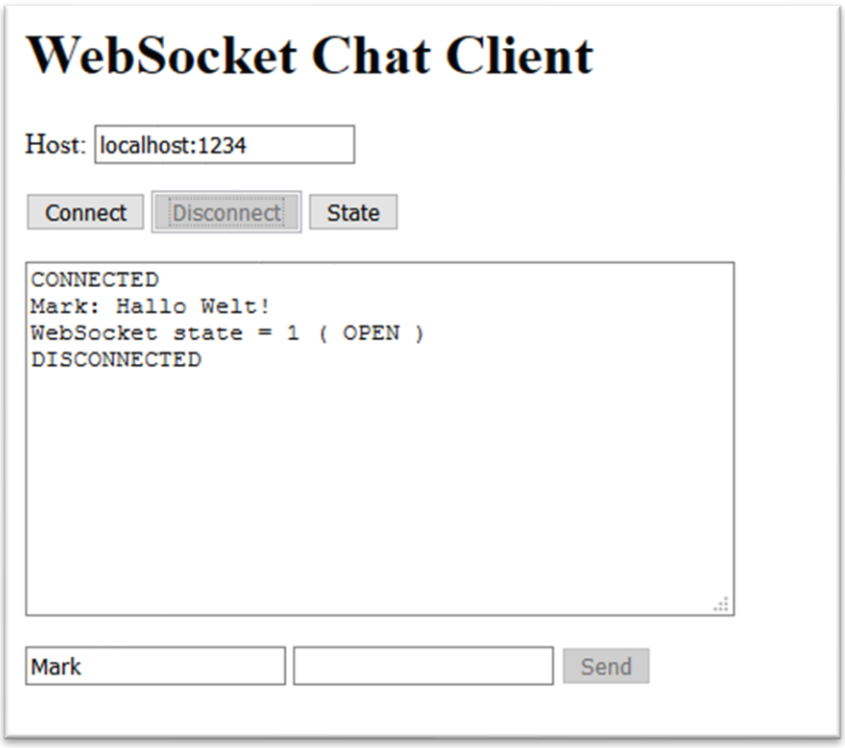
FH-OÖ Hagenberg/HSD
Embedded Visualization with Qt
Übung 1 Simplechat Client (_____ / 24 Punkte)

Raspberry Pi SimpleChat

Ziel: Umsetzung eines SimpleChat Clients für den RaspberryPI mit QWidgets

Folgender Screenshot des SimpleChat-Web-Clients soll sinngemäß auch am RaspberryPI funktionieren:



Erstelle einen lauffähigen Prototyp, der optimiert auf die Platzverhältnisse des Raspberry Pi Bildschirmes ist (800x480):

- Die Elemente im Screenshot oben sind nicht optimal angeordnet. Versuche mit QLayouts eine gute Anordnung der Elemente zu erstellen und somit auch eine flexible Darstellung der Elemente zu erreichen.
- Da der Chat-Client als Embedded-Anwendung ausgerichtet sein soll, ist es nicht notwendig Fensterrahmen oder Menüleisten anzuzeigen. Der Inhalt des Bildschirmes soll vollflächig mit den dargestellten Inhalten genutzt werden. (Tip: Methode showFullScreen() nutzen)
- Nutze setStyleSheet() um auf den Buttons zu visualisieren, ob eine aktive Verbindung (=grün) besteht oder keine Verbindung (=Default-Farbe des buttons) besteht.
- Nutze Stylesheets oder QFont um die Schriftgrößen auf den Buttons und Labels und den Eingabefeldern zu setzen. Wichtig ist eine gute Lesbarkeit am RaspberryPI.
- Unterteile die Anwendung in sinnvolle Klassen: Die GUI sollte in einer separaten Klasse abgedeckt werden. Die ChatClient-Komponente soll in einer eigenen Klasse abgedeckt werden.

Hinweise für die Abgabe

Im abgegebenen gezippten File soll folgender Inhalt enthalten sein:

- Qt Namenskonventionen und coding style beachten
- Source-Code-Files inklusive Qt-Project File und PDF als ZIP
- Das PDF soll so aufgebaut sein:
    - Übungsangabe als Deckblatt
    - Ausgearbeitete Lösungsidee, Screenshot von der ausgeführten Raspberry-Pi-Software.
        - Hinweis: Screenshots am Raspberry-Pi können mit dem Smartphone gemacht werden.
    - Sämtlicher Source Code in Text-Form: Qt-Project-File, main.cpp, …
- Füge der Lösungsidee auch einen Screenshot hinzu, die am Ubuntu Desktop gemacht wird und demonstriert, wie die Anwendung bei unterschiedlichen Fenstergrößen aussieht.

Inhaltsverzeichnis

Lösungsidee

Es werden insgesamt zwei Klassen erstellt, eine verwaltet das GUI, die andere die Verbindung mit dem Server. In der GUI-Verwaltung werden im Konstruktor sämtliche benötigten Komponenten erzeugt. Die Komponenten welche nebeneinander liegen werden in ein HBoxLayout gelegt, diese HBoxLayouts und die restlichen Komponenten anschließend in ein VBoxLayout. Weiters muss ein Member eine Instanz der Verbindungsklasse beinhalten. So können nun Eventhandler für die clicked-Signale aller Buttons geschrieben werden. Anschließend müssen noch Slots für das aufbauen und abbauen der Verbindung sowie das erhalten einer Nachricht zur Verfügung gestellt werden.
Die Verbindungsklasse kapselt nun im Prinzip nur ein QWebSocket-Objekt, um dieses mit der GUI-Klasse zu verbinden.

Sourcecode

```
QT       += core gui
QT       += websockets

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 += \
    chatclientwebconnect.cpp \
    main.cpp \
    simplechatclient.cpp

HEADERS += \
    chatclientwebconnect.h \
    simplechatclient.h

FORMS +=

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

Listing 2.1: Project-File

```
#include "simplechatclient.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    SimpleChatClient w;
    w.show();
    return a.exec();
}
```

Listing 2.2: main-File

```
#ifndef SIMPLECHATCLIENT_H
#define SIMPLECHATCLIENT_H

#include <QMainWindow>
#include <QLabel>
#include <QTextEdit>
#include <QPushButton>
#include <QLineEdit>
#include "chatclientwebconnect.h"

class SimpleChatClient : public QMainWindow
{
    Q_OBJECT

public:
    SimpleChatClient(QWidget *parent = nullptr);
    ~SimpleChatClient();
public slots:
    void onConnectButtonClicked();
    void onDisconnectButtonClicked();
    void onSendButtonClicked();
    void onStateButtonClicked();

private:
    chatclientwebconnect *m_chatClient;
    QLabel* headerLabel;
    QLabel* hostLabel;
    QLineEdit* serverAddressLineEdit;
    QTextEdit* chatWindowTextEdit;
    QLineEdit* usernameLineEdit;
    QLineEdit* messageLineEdit;
    QPushButton* connectPushButton;
    QPushButton* disconnectPushButton;
    QPushButton* statePushButton;
    QPushButton* sendPushButton;

    void handleConnect();
    void handleDisconnect();
    void handleMessageReceived(QString message);


```

```
42 };
43 #endif // SIMPLECHATCLIENT_H
```

Listing 2.3: Header der GUI Verwaltung

```
#include "simplechatclient.h"

#include <QVBoxLayout>

SimpleChatClient::SimpleChatClient(QWidget *parent)
    : QMainWindow(parent)
{
    m_chatClient = new chatclientwebconnect(this);

    //create widgets
    this->headerLabel=new QLabel("WebSocket Chat Client");
    this->hostLabel=new QLabel ("Host:") ;
    this->serverAddressLineEdit=new QLineEdit();
    this->chatWindowTextEdit=new QTextEdit();
    this->usernameLineEdit=new QLineEdit();
    this->messageLineEdit=new QLineEdit();
    this->connectPushButton=new QPushButton("Connect");
    this->disconnectPushButton=new QPushButton("Disconnect");
    this->statePushButton=new QPushButton("State");
    this->sendPushButton=new QPushButton("Send");

    //positioning
    QVBoxLayout* mainLayout = new QVBoxLayout();
    QHBoxLayout* hostLayout = new QHBoxLayout();
    QHBoxLayout* connectButtonsLayout = new QHBoxLayout();
    QHBoxLayout* messageLayout = new QHBoxLayout();

    hostLayout->addWidget(hostLabel);
    hostLayout->addWidget(serverAddressLineEdit);

    connectButtonsLayout->addWidget(connectPushButton);
    connectButtonsLayout->addWidget(disconnectPushButton);
    connectButtonsLayout->addWidget(statePushButton);

    messageLayout->addWidget(usernameLineEdit);
    messageLayout->addWidget(messageLineEdit);
    messageLayout->addWidget(sendPushButton);

    mainLayout->addWidget(headerLabel);
    mainLayout->addLayout(hostLayout);
    mainLayout->addLayout(connectButtonsLayout);
```

```
42      mainLayout->addWidget(chatWindowTextEdit);
43      mainLayout->addLayout(messageLayout);
44
45      QWidget* mainWidget = new QWidget();
46      mainWidget->setLayout(mainLayout);
47      this->setCentralWidget(mainWidget);
48
49      //websocket functionality
50      connect(m_chatClient,&chatclientwebconnect::connectedToServer, this, &
            SimpleChatClient::handleConnect);
51      connect(m_chatClient,&chatclientwebconnect::disconnectedFromServer, this, &
            SimpleChatClient::handleDisconnect);
52      connect(m_chatClient,&chatclientwebconnect::messageReceived, this, &
            SimpleChatClient::handleMessageReceived);
53
54      //connect button signals
55      connect(this->connectPushButton,&QPushButton::clicked,this, &SimpleChatClient::
            onConnectButtonClicked);
56      connect(this->disconnectPushButton, &QPushButton::clicked,this,&SimpleChatClient
            ::onDisconnectButtonClicked);
57      connect(this->statePushButton, &QPushButton::clicked, this, &SimpleChatClient::
            onStateButtonClicked);
58      connect(this->sendPushButton, &QPushButton::clicked, this, &SimpleChatClient::
            onSendButtonClicked);
59
60      this->headerLabel->setStyleSheet("font-weight: bold; font-size: 17px");
61      this->setStyleSheet("QLineEdit { font-size: 15px }");
62      this->setStyleSheet("QPushButton { font-size: 16px}");
63
64      //initial state of buttons
65      this->disconnectPushButton->setEnabled(false);
66      this->connectPushButton->setEnabled(true);
67      this->connectPushButton->setStyleSheet("background-color:red");
68
69      this->serverAddressLineEdit->setText("ws://localhost:1234");
70      this->usernameLineEdit->setText("Riegler");
71
72      this->chatWindowTextEdit->setReadOnly(true);
73
74      //this->showFullScreen();
75      qDebug() << "SimpleChatClient constructor done";
```

```
76 }
77
78 void SimpleChatClient::handleConnect(){
79     qDebug() << "handleConnect called";
80     this->connectPushButton->setEnabled(false);
81     this->disconnectPushButton->setEnabled(true);
82     this->connectPushButton->setStyleSheet("background-color:green");
83     this->chatWindowTextEdit->append("CONNECTED");
84 }
85
86 void SimpleChatClient::handleDisconnect(){
87     qDebug() << "handleDisconnect called";
88     this->disconnectPushButton->setEnabled(false);
89     this->connectPushButton->setEnabled(true);
90     this->connectPushButton->setStyleSheet("background-color:red");
91     this->chatWindowTextEdit->append("DISCONNECTED");
92 }
93
94 void SimpleChatClient::handleMessageReceived(QString message){
95     qDebug() << "handleMessageReceived called";
96     this->chatWindowTextEdit->append(message);
97 }
98
99 void SimpleChatClient::onConnectButtonClicked(){
100     qDebug() << "connectButtonClicked";
101     QString host = this->serverAddressLineEdit->text();
102     if(host!=nullptr){
103         m_chatClient->connectToServer(host);
104     }else{
105         this->chatWindowTextEdit->append("No server entered");
106     }
107
108 }
109 void SimpleChatClient::onDisconnectButtonClicked(){
110     qDebug() << "disconnectButtonClicked";
111     m_chatClient->disconnectFromServer();
112 }
113 void SimpleChatClient::onSendButtonClicked(){
114     qDebug() << "sendButtonClicked";
115     QString message = this->messageLineEdit->text();
116     QString name = this->usernameLineEdit->text();
```

```
117     QString sendstring;
118     if(name != nullptr){
119         sendstring += name;
120     }else{
121         sendstring += "<empty>";
122     }
123     sendstring += ": ";
124     if(message != nullptr){
125         sendstring += message;
126     }
127     m_chatClient->sendMessage(sendstring);
128     this->chatWindowTextEdit->append(sendstring);
129     //clear input
130     this->messageLineEdit->setText("");
131 }
132 void SimpleChatClient::onStateButtonClicked(){
133     qDebug() << "stateButtonClicked";
134     this->chatWindowTextEdit->append(this->m_chatClient->getConnectionState());
135 }
136
137 SimpleChatClient::~SimpleChatClient()
138 {
139 }
```

Listing 2.4: Implementierung der GUI Verwaltung

```
#ifndef CHATCLIENTWEBCONNECT_H
#define CHATCLIENTWEBCONNECT_H


#include <QtCore>
#include <QtCore/QObject>
#include <QtWebSockets>


class chatclientwebconnect : public QObject
{
    Q_OBJECT
public:
    chatclientwebconnect(QObject* parent);

    void connectToServer(QString url);
    void onConnected();
    void onDisconnected();
    void onMessageReceived(QString message);
    void disconnectFromServer();
    void sendMessage(QString message);
    QString getConnectionState();
    virtual ~chatclientwebconnect() {}
private:
    QWebSocket m_qchatWebSocket;
    bool m_isConnected;
signals:
    void connectedToServer();
    void disconnectedFromServer();
    void messageReceived(QString message);
};

#endif // CHATCLIENTWEBCONNECT_H
```

Listing 2.5: Header der Verbindungsverwaltung

```cpp
#include "chatclientwebconnect.h"

#include <QtWebSockets>
#include <QtCore>

chatclientwebconnect::chatclientwebconnect(QObject* parent) : QObject(parent){
    //init signals for connect/disconnect from server
    //message received can only be set after a server was connected
    connect(&m_qchatWebSocket,&QWebSocket::connected,this,&chatclientwebconnect::
        onConnected);
    connect(&m_qchatWebSocket, &QWebSocket::disconnected, this, &chatclientwebconnect
        ::onDisconnected);
}

void chatclientwebconnect::connectToServer(QString url){
    m_qchatWebSocket.open(QUrl(url));
    qDebug() << "opened connection to " + url;
}

void chatclientwebconnect::onConnected(){
    m_isConnected = true;
    //callback for successful connection
    emit connectedToServer();
    connect(&m_qchatWebSocket, &QWebSocket::textMessageReceived,this,&
        chatclientwebconnect::onMessageReceived);
    qDebug() << "server connected";
}

void chatclientwebconnect::onDisconnected(){
    qDebug() << "server disconnected";
    emit disconnectedFromServer();
}

void chatclientwebconnect::onMessageReceived(QString message){
    emit messageReceived(message);
    qDebug() << "Received message: " << message;
}

void chatclientwebconnect::disconnectFromServer(){
    m_qchatWebSocket.close();
    qDebug() << "closed connection";
```

```
39 }
40
41 void chatclientwebconnect::sendMessage(QString message){
42     qDebug() << "Sent message: " << message;
43     m_qchatWebSocket.sendTextMessage(message);
44 }
45 QString chatclientwebconnect::getConnectionState(){
46     switch(m_qchatWebSocket.state()){
47     case QAbstractSocket::UnconnectedState:
48         return "Unconnected";
49     case QAbstractSocket::HostLookupState:
50         return "host lookup";
51     case QAbstractSocket::ConnectingState:
52         return "Connecting";
53     case QAbstractSocket::ConnectedState:
54         return "Connected";
55     case QAbstractSocket::BoundState:
56         return "Bound";
57     case QAbstractSocket::ListeningState:
58         return "Listening";
59     case QAbstractSocket::ClosingState:
60         return "Closing";
61     default:
62         return "Invalid";
63     }
64 }
```

Listing 2.6: Implementierung der Verbindungsverwaltung

Testfälle

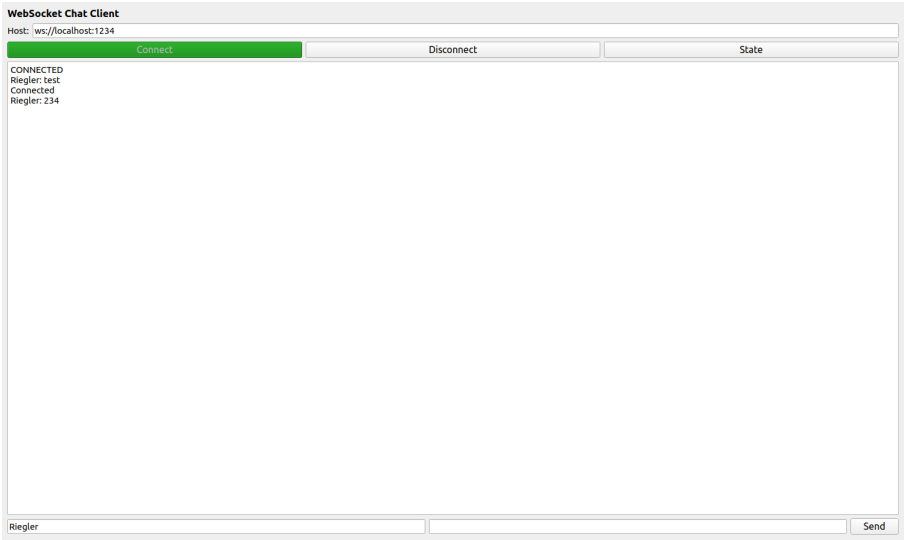


Abbildung 3.1: Funktionstest Ubuntu

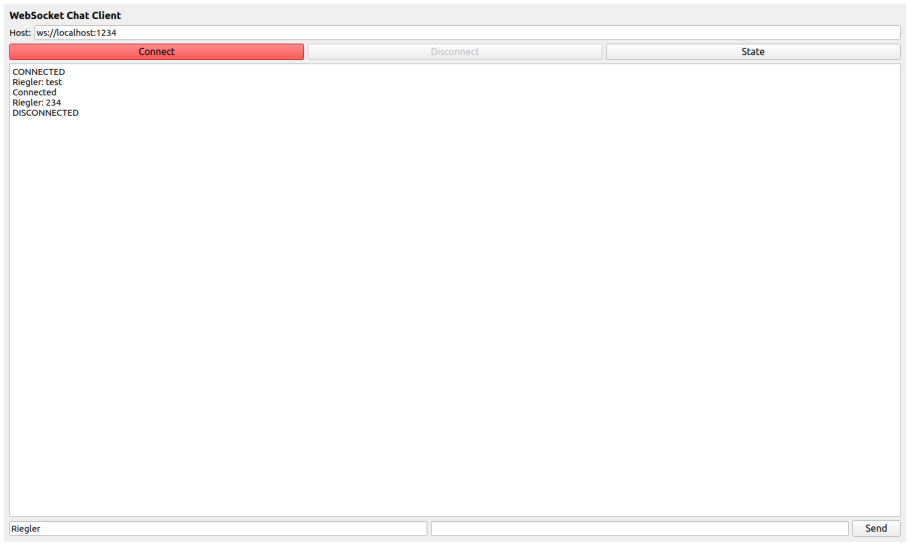


Abbildung 3.2: Funktionstest Ubuntu

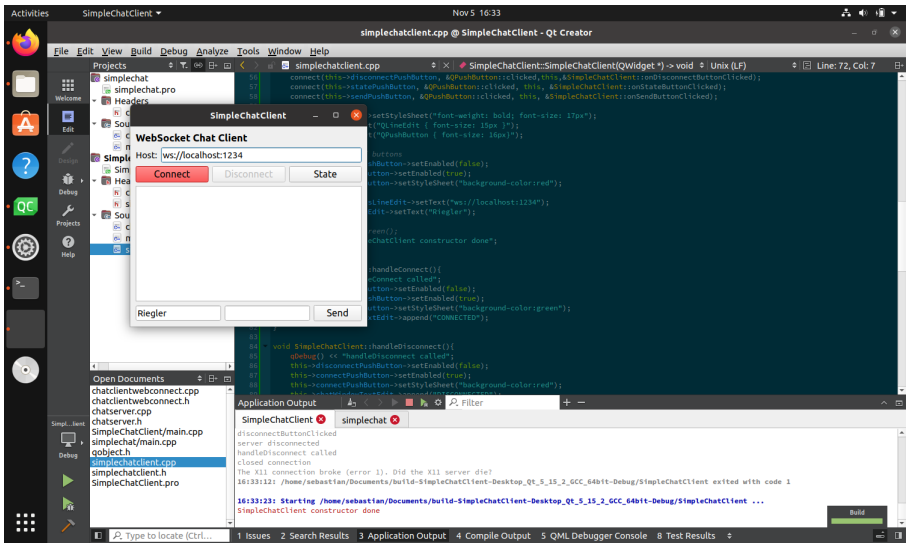


Abbildung 3.3: Layouttest Ubuntu

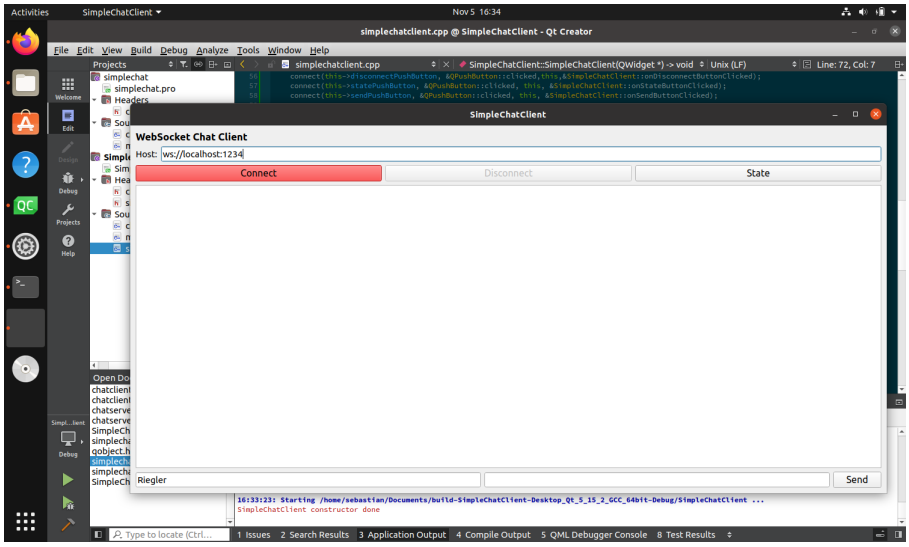


Abbildung 3.4: Layouttest Ubuntu

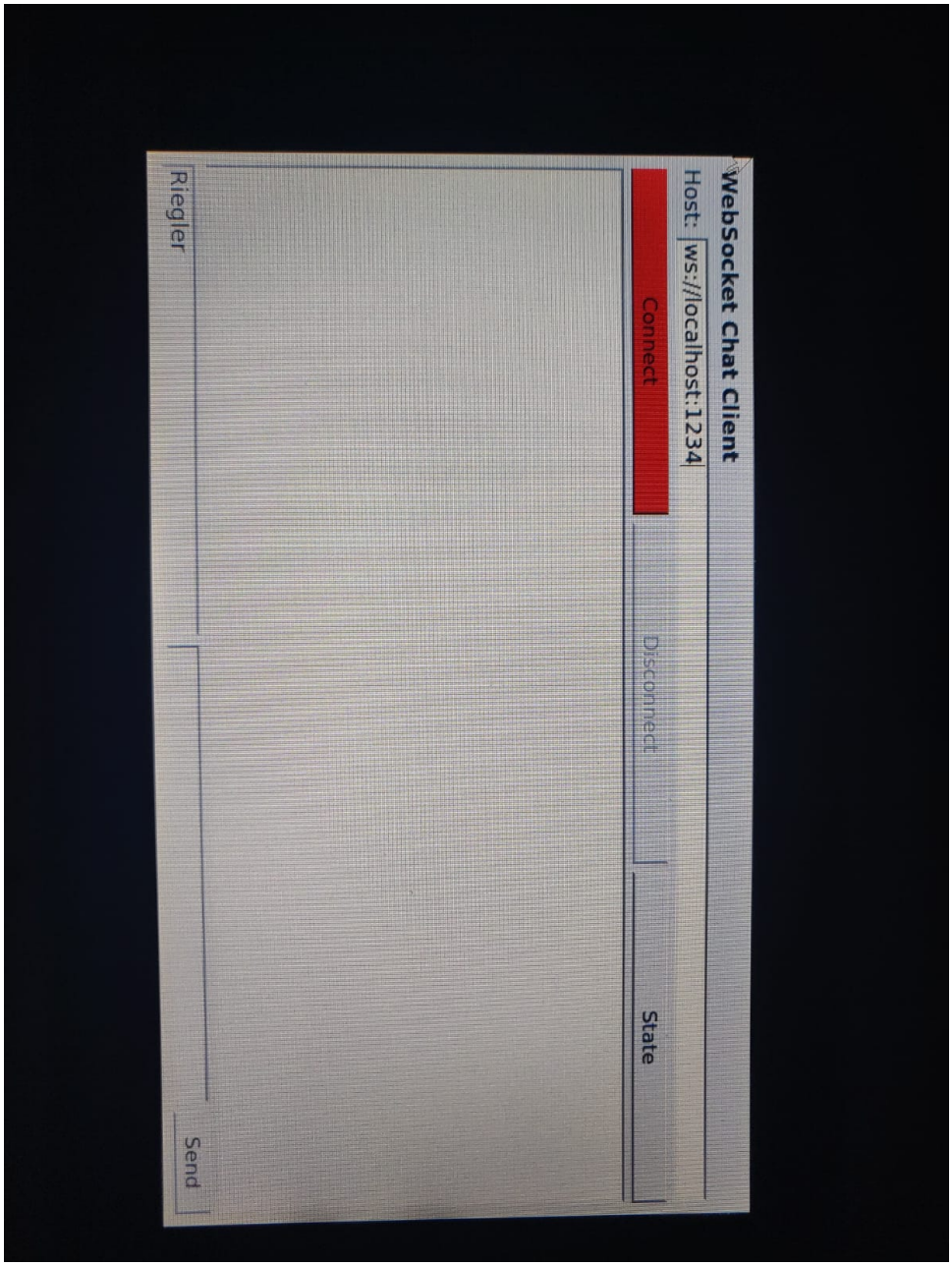


Abbildung 3.5: Layouttest Raspi

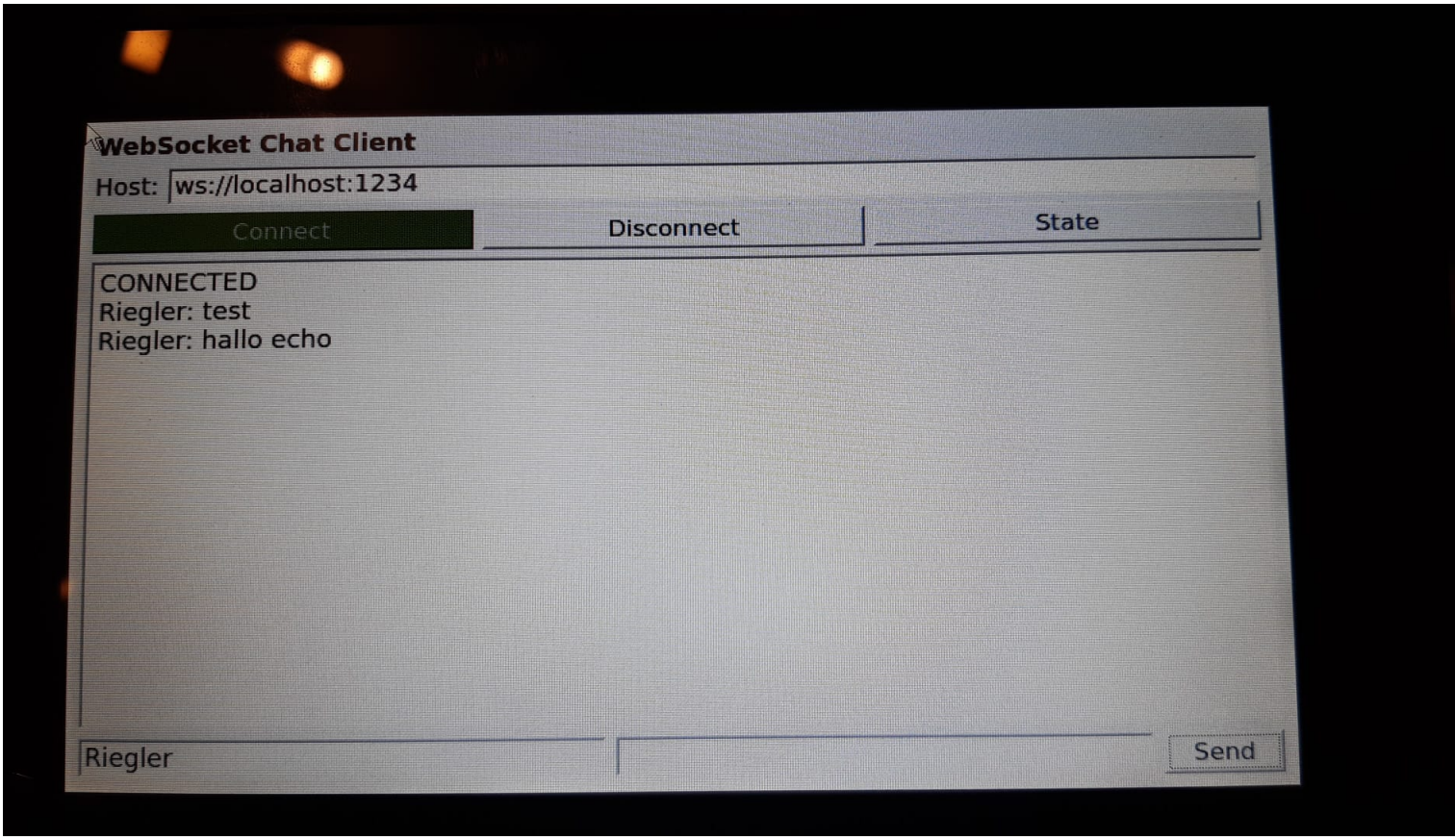


Abbildung 3.6: Funktionstest Raspi