AfficheurServiceWebsocket.pro

```
QT -= gui
QT += websockets

CONFIG += c++11 console
CONFIG -= app_bundle

TARGET = serviceAfficheur
target.path = /home/pi/c++
INSTALLS += target

HEADERS += serialib.h afficheur.h service.h
SOURCES += main.cpp serialib.cpp afficheur.cpp service.cpp
```

main.cpp 1

```
#include <QCoreApplication>
#include <service.h>

int main(int argc, char *argv[])
{
  QCoreApplication a(argc, argv);
  Service service;
  Service::connect(&service, SIGNAL(stop()), &a, SLOT(quit()));
  return a.exec();
}
```

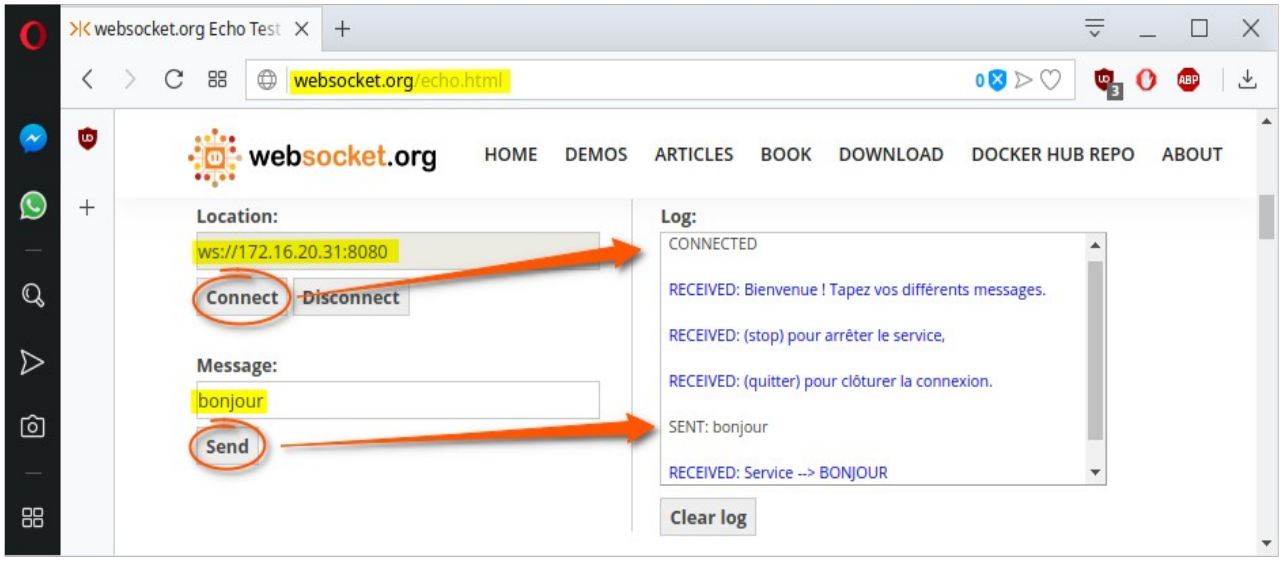
afficheur.h 1

```
#ifndef AFFICHEUR_H
#define AFFICHEUR_H

#include "serialib.h"
#include <string>
using namespace std;

class Afficheur
{
  serialib usb;
  bool connexion;
public:
  Afficheur()   { connexion = usb.Open("/dev/ttyUSB0", 9600) == 1; }
  ~Afficheur()  { usb.Close(); }
  bool etat() const  { return connexion; }
  void affichePage(const string &message);
private:
  string checksum(string texte);
};

#endif // AFFICHEUR_H
```



```cpp
#include "afficheur.h"
#include <sstream>

void Afficheur::affichePage(const string &message)
{
  string commande = "<L1><PA><FE><MA><WC><FE><CE>";
  commande += message;
  ostringstream trame;
  trame << "<ID01>" << commande << checksum(commande) << "<E>";
  usb.WriteString(trame.str().c_str());
  usleep(50000);
}

string Afficheur::checksum(string texte)
{
  char calcul = 0;
  for (char octet : texte) calcul^=octet;
  char bas = calcul & 0b00001111;
  char haut = (calcul & 0b11110000) >> 4;
  bas = bas < 10 ? bas+=0x30 :bas+=0x41-10;
  haut = haut < 10 ? haut+=0x30 : haut+=0x41-10;
  return {haut, bas};
}
```



```cpp
#ifndef SERVICE_H
#define SERVICE_H

#include <QObject>
#include <QWebSocketServer>
#include <afficheur.h>

class Service : public QObject
{
  Q_OBJECT
public:
  explicit Service(QObject *parent = nullptr);
signals:
  void stop();
private slots:
  void nouvelleConnexion();
  void receptionMessage(const QString& message);
  void deconnexion();
private:
  QWebSocketServer service;
  Afficheur lcd;
};

#endif // SERVICE_H
```

```
#include "service.h"
#include <QWebSocket>

Service::Service(QObject *parent) : QObject(parent),
  service("afficheur", QWebSocketServer::NonSecureMode, this)
{
  if (lcd.etat())
    if (service.listen(QHostAddress::Any, 8080)) {
      connect(&service, SIGNAL(newConnection()), this,
SLOT(nouvelleConnexion()));
      lcd.affichePage("Service actif");
    }
  else stop();
}

void Service::nouvelleConnexion()
{
  QWebSocket* client = service.nextPendingConnection();
  connect(client, SIGNAL(textMessageReceived(QString)), this,
SLOT(receptionMessage(QString)));
  connect(client, SIGNAL(disconnected()), this, SLOT(deconnexion()));
  client->sendTextMessage("Bienvenue !");
}

void Service::receptionMessage(const QString &message)
{
  if (message=="stop") {
    lcd.affichePage("Service inactif!");
    stop();
  }
  else lcd.affichePage(message.toStdString());
}

void Service::deconnexion()
{
  QWebSocket* client = (QWebSocket*) sender();
  disconnect(client, SIGNAL(textMessageReceived(QString)), this,
SLOT(receptionMessage(QString)));
  disconnect(client, SIGNAL(disconnected()), this, SLOT(deconnexion()));
}
```



```
QT += quick websockets
CONFIG += c++11
TARGET = Afficheur

DEFINES += QT_DEPRECATED_WARNINGS

SOURCES += main.cpp controleur.cpp
HEADERS += controleur.h

RESOURCES += qml.qrc

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

```cpp
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <controleur.h>
#include <QtQml>

int main(int argc, char *argv[])
{
  QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
  QGuiApplication app(argc, argv);

  Controleur controleur;
  QQmlApplicationEngine engine;

  engine.rootContext()->setContextProperty("controleur", &controleur);

  engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
  if (engine.rootObjects().isEmpty()) return -1;

  return app.exec();
}
```



```cpp
#ifndef CONTROLEUR_H
#define CONTROLEUR_H

#include <QObject>
#include <QWebSocket>

class Controleur : public QObject
{
  Q_OBJECT
  Q_PROPERTY(QString message READ getMessage NOTIFY messageChanged)
  Q_PROPERTY(QString adresse MEMBER adresse NOTIFY adresseChanged)

public:
  explicit Controleur(QObject *parent = nullptr) : QObject(parent) {}
  QString getMessage() const { return message; }

signals:
  void messageChanged();
  void adresseChanged();
public slots:
  void soumettre(const QString& texte);
  void reception(const QString& texte);
  void connexion();
  void deconnexion();
private:
  QWebSocket service;
  QString adresse="172.16.2.35", message;
  int port=8080;
};

#endif // CONTROLEUR_H
```

controleur.cpp 1

```cpp
#include "controleur.h"

void Controleur::connexion()
{
  connect(&service,SIGNAL(textMessageReceived(QString)),this,SLOT(reception(QString)));
  service.open(QUrl(QString("ws://%1:%2").arg(adresse).arg(port)));
}

void Controleur::soumettre(const QString &texte)
{
  service.sendTextMessage(texte);
}

void Controleur::reception(const QString &texte)
{
  message = texte;
  messageChanged();
}

void Controleur::deconnexion()
{
  service.close();
  message = "Connexion interrompue !";
  messageChanged();
}
```

main.qml 1

```qml
import QtQuick 2.9
import QtQuick.Controls 2.4

ApplicationWindow {
  visible: true
  width: 320
  height: 480
  color: "purple"
  title: qsTr("Afficheur AM-03127-LED")

  TextField {
    id: adresse
    anchors {
      top: parent.top
      topMargin: 8
      left: parent.left
      leftMargin: 8
      right: connexion.left
      rightMargin: 8
    }
    color: "purple"
    text: controleur.adresse
    inputMethodHints: Qt.ImhDigitsOnly
  }

  ToolButton {
    id: connexion
    anchors {
      top: parent.top
      topMargin: 8
      right: parent.right
      rightMargin: 8
    }
    text: qsTr("Connexion")
    checkable: true
```

```
    onCheckedChanged: {
      if (checked) {
        controleur.adresse = adresse.text
        controleur.connexion()
        text = "Deconnexion"
        envoi.enabled = true
      }
      else {
        controleur.deconnexion()
        text = "Connexion"
        envoi.enabled = false
      }
    }
  }

  TextField {
    id: envoi
    anchors {
      top: adresse.bottom
      topMargin: 8
      left: parent.left
      leftMargin: 8
      right: parent.right
      rightMargin: 8
    }
    placeholderText: qsTr("Message à envoyer")
    color: "purple"
    onAccepted: {
      controleur.soumettre(text)
      selectAll()
    }
    enabled: false
  }

  TextArea {
    id: réponse
    anchors.centerIn: parent
    width: 220
    height: 100
    font.pixelSize: 16
    background: Rectangle {
      color: "lavender"
      radius: 10
    }
    color: "purple"
    readOnly: true
    wrapMode: "Wrap"
    text: "Connectez-vous"
    OpacityAnimator on opacity {
      id: alerte
      from: 1
      to: 0
      duration: 4000
      easing.type: Easing.InQuart
    }
  }

  Connections {
    target: controleur
    onMessageChanged: {
      alerte.start()
      réponse.text = controleur.message
    }
  }
}
```