notes-websocket-json.pro 1

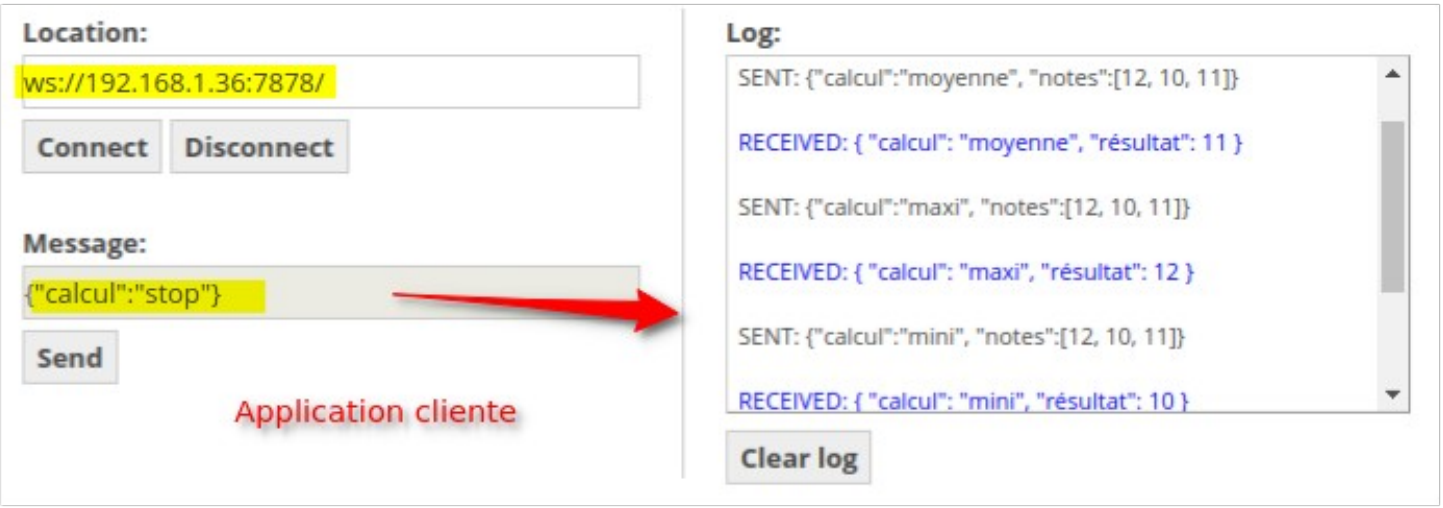
```
QT -= gui
QT += websockets

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

DEFINES += QT_DEPRECATED_WARNINGS
HEADERS += notes.h
SOURCES += main.cpp notes.cpp

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

main.cpp 1

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

int main(int argc, char *argv[])
{
  QCoreApplication a(argc, argv);
  // Création de l'objet contenant le service de calcul
  Notes notes;
  // Arrêt de l'application dès que le signal "stop" du service est lancé
  QObject::connect(&notes, SIGNAL(stop()), &a, SLOT(quit()));
  return a.exec();
}
```

notes.h 1

```
#ifndef NOTES_H
#define NOTES_H

#include <QObject>
#include <QWebSocketServer>

class Notes : public QObject
{
  Q_OBJECT
public:
  explicit Notes(QObject *parent = nullptr);
signals:
  void stop(); // Arrêt définitive du service
private slots:
  void nouvelleConnexion(); // Un nouveau client se connecte
  void receptionRequete(const QString& requete); // Demande de réaliser une ensemble de calculs
  void deconnexion(); // Un client se déconnecte effectivement
private:
  QWebSocketServer service; // Objet représentant le service
};

#endif // NOTES_H
```



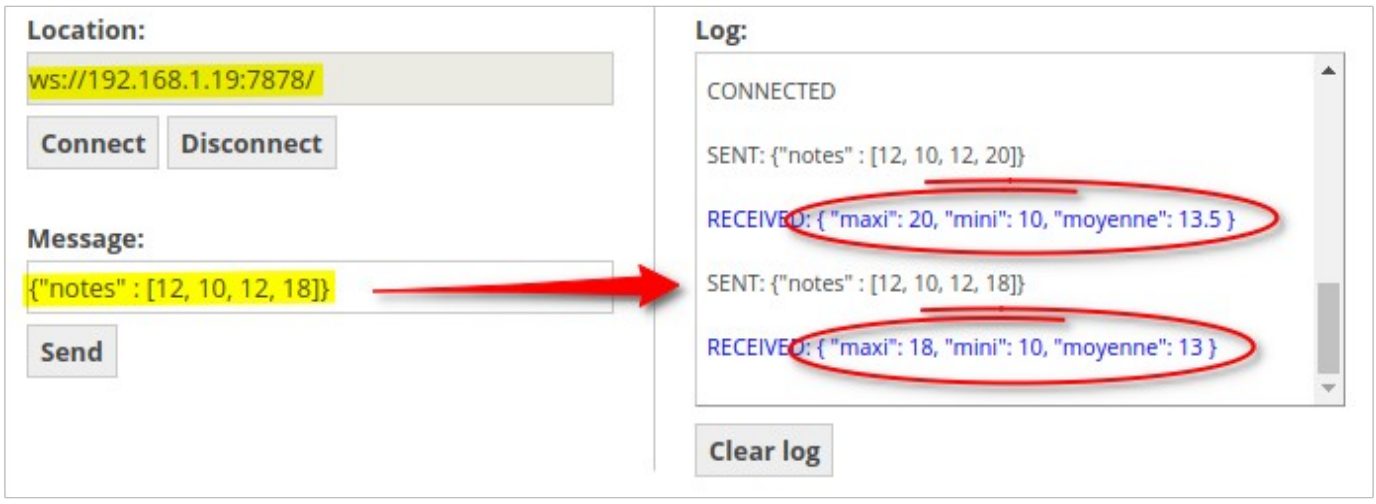
```cpp
#include "notes.h"
#include <QWebSocket>
#include <QJsonObject>
#include <QJsonArray>
#include <QJsonDocument>
#include <QDebug> // Permet de tracer les événements dans la console du service

Notes::Notes(QObject *parent) : QObject(parent), service("notes", QWebSocketServer::NonSecureMode, this)
{
  if (service.listen(QHostAddress::Any, 7878))
  {
    connect(&service, SIGNAL(newConnection()), this, SLOT(nouvelleConnexion()));
    qDebug("Démarrage du service 7878...");
  }
}

void Notes::nouvelleConnexion()
{
  QWebSocket *client = service.nextPendingConnection();
  connect(client, SIGNAL(textMessageReceived(QString)), this, SLOT(receptionRequete(QString)));
  connect(client, SIGNAL(disconnected()), this, SLOT(deconnexion()));
  qDebug() << "Nouveau client connecté : " << client->peerAddress().toString();
}

void Notes::receptionRequete(const QString &requete)
{
  qDebug() << "Récupération requête :" << requete;
  QWebSocket *client = (QWebSocket *) sender();
  QJsonDocument document = QJsonDocument::fromJson(requete.toUtf8());
  qDebug() << "Document :" << document.toJson();
  QJsonObject json = document.object();
  QString calcul = json["calcul"].toString();
  QJsonArray notes = json["notes"].toArray();
  QJsonObject resultat;
  resultat["calcul"] = json["calcul"];
  if (calcul=="moyenne") {
    double somme = 0.0;
    int nombre = notes.size();
    for (QJsonValue note : notes) somme += note.toDouble();
    resultat["résultat"] = somme/nombre;
  }
  else if (calcul=="maxi") {
    double maxi=0.0;
    for (QJsonValue note : notes)
      if (note.toDouble()>maxi) maxi=note.toDouble();
    resultat["résultat"] = maxi;
  }
  else if (calcul=="mini") {
    double mini=20.0;
    for (QJsonValue note : notes)
      if (note.toDouble()<mini) mini=note.toDouble();
    resultat["résultat"] = mini;
  }
  else if (calcul=="stop") { stop(); return; }
  QJsonDocument reponse(resultat);
  qDebug() << "Résultat :" << reponse.toJson();
  client->sendTextMessage(reponse.toJson());
}

void Notes::deconnexion()
{
  QWebSocket *client = (QWebSocket *) sender();
  client->deleteLater();
  disconnect(client, SIGNAL(textMessageReceived(QString)), this, SLOT(receptionRequete(QString)));
  disconnect(client, SIGNAL(disconnected()), this, SLOT(deconnexion()));
  qDebug() << "Déconnexion du client : " << client->peerAddress().toString();
}
```

Fichier Édition Affichage Rechercher Terminal Aide

```
Démarrage du service 7878...
Nouveau client connecté :  "::ffff:192.168.1.36"
Récupération requête : "{\"calcul\":\"moyenne\", \"notes\":[12, 10, 11]}"
Document : "{\n    \"calcul\": \"moyenne\",\n    \"notes\": [\n        12,\n
    10,\n        11\n    ]\n}\n"
Résultat : "{\n    \"calcul\": \"moyenne\",\n    \"r\xC3\xA9sultat\": 11\n}\n"
Récupération requête : "{\"calcul\":\"maxi\", \"notes\":[12, 10, 11]}"
Document : "{\n    \"calcul\": \"maxi\",\n    \"notes\": [\n        12,\n
 10,\n        11\n    ]\n}\n"
Résultat : "{\n    \"calcul\": \"maxi\",\n    \"r\xC3\xA9sultat\": 12\n}\n"
Récupération requête : "{\"calcul\":\"mini\", \"notes\":[12, 10, 11]}"
Document : "{\n    \"calcul\": \"mini\",\n    \"notes\": [\n        12,\n
 10,\n        11\n    ]\n}\n"
Résultat : "{\n    \"calcul\": \"mini\",\n    \"r\xC3\xA9sultat\": 10\n}\n"
Récupération requête : "{\"calcul\":\"stop\"}"
Document : "{\n    \"calcul\": \"stop\"\n}\n"
Déconnexion du client :  "::ffff:192.168.1.36"
Appuyez sur <ENTRÉE> pour fermer cette fenêtre...
```

Traces côté serveur

```cpp
#include "notes.h"
#include <QWebSocket>
#include <QJsonObject>
#include <QJsonArray>
#include <QJsonDocument>
#include <QDebug> // Permet de tracer les événements dans la console du service

Notes::Notes(QObject *parent) : QObject(parent), service("notes", QWebSocketServer::NonSecureMode, this)
{
  if (service.listen(QHostAddress::Any, 7878))
  {
    connect(&service, SIGNAL(newConnection()), this, SLOT(nouvelleConnexion()));
    qDebug("Démarrage du service 7878...");
  }
}

void Notes::nouvelleConnexion()
{
  QWebSocket *client = service.nextPendingConnection();
  connect(client, SIGNAL(textMessageReceived(QString)), this, SLOT(receptionRequete(QString)));
  connect(client, SIGNAL(disconnected()), this, SLOT(deconnexion()));
  qDebug() << "Nouveau client connecté : " << client->peerAddress().toString();
}

void Notes::receptionRequete(const QString &requete)
{
  qDebug() << "Récupération requête :" << requete;
  QWebSocket *client = (QWebSocket *) sender();
  QJsonDocument document = QJsonDocument::fromJson(requete.toUtf8());
  qDebug() << "Document :" << document.toJson();
  QJsonObject json = document.object();
  QJsonArray notes = json["notes"].toArray();
  double somme = 0.0;
  int nombre = notes.size();
  double maxi = 0.0;
  double mini = nombre>0 ? 20.0 : 0.0;
  for (QJsonValue note : notes) {
    somme += note.toDouble();
    if (note.toDouble()>maxi) maxi=note.toDouble();
    if (note.toDouble()<mini) mini=note.toDouble();
  }
  QJsonObject resultat;
  resultat["moyenne"] = somme/nombre;
  resultat["maxi"] = maxi;
  resultat["mini"] = mini;
  QJsonDocument reponse(resultat);
  qDebug() << "Résultat :" << reponse.toJson();
  client->sendTextMessage(reponse.toJson());
}

void Notes::deconnexion()
{
  QWebSocket *client = (QWebSocket *) sender();
  client->deleteLater();
  disconnect(client, SIGNAL(textMessageReceived(QString)), this, SLOT(receptionRequete(QString)));
  disconnect(client, SIGNAL(disconnected()), this, SLOT(deconnexion()));
  qDebug() << "Déconnexion du client : " << client->peerAddress().toString();
}
```



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

DEFINES += QT_DEPRECATED_WARNINGS
HEADERS += controleur.h
SOURCES += controleur.cpp main.cpp
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
```



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

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

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

  const QUrl url(QStringLiteral("qrc:/main.qml"));
  QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
                   &app, [url](QObject *obj, const QUrl &objUrl) {
    if (!obj && url == objUrl)
      QCoreApplication::exit(-1);
  }, Qt::QueuedConnection);
  engine.load(url);

  return app.exec();
}
```



```
#ifndef CONTROLEUR_H
#define CONTROLEUR_H

#include <QObject>
#include <QWebSocket>

class Controleur : public QObject
{
  Q_OBJECT
  Q_PROPERTY(QStringList notes MEMBER notes NOTIFY notesChanged)
  Q_PROPERTY(double moyenne MEMBER moyenne)
  Q_PROPERTY(double maxi MEMBER maxi)
  Q_PROPERTY(double mini MEMBER mini)
public:
  explicit Controleur(QObject *parent = nullptr);
signals:
  void notesChanged();
  void valeursChanged();
public slots:
  void ajouterNote(const QString& note);
  void supprimer(const QString& note);
  void reponse(const QString& texte);
private:
  void requete();
private:
  QStringList notes;
  double moyenne, maxi, mini;
  QWebSocket service;
};

#endif // CONTROLEUR_H
```



```
#include "controleur.h"
#include <QJsonObject>
#include <QJsonArray>
#include <QJsonDocument>

Controleur::Controleur(QObject *parent) : QObject(parent)
{
  connect(&service, SIGNAL(textMessageReceived(QString)), this, SLOT(reponse(QString)));
  service.open(QUrl(QString("ws://192.168.1.19:7878/")));
}
```

```cpp
void Controleur::ajouterNote(const QString &note)
{
  notes << note;
  notesChanged();
  requete();
}

void Controleur::supprimer(const QString &note)
{
  notes.removeOne(note);
  notesChanged();
  requete();
}

void Controleur::reponse(const QString &texte)
{
  QJsonDocument document = QJsonDocument::fromJson(texte.toUtf8());
  QJsonObject json = document.object();
  moyenne = json["moyenne"].toDouble();
  maxi = json["maxi"].toDouble();
  mini = json["mini"].toDouble();
  valeursChanged();
}

void Controleur::requete()
{
  QJsonObject json;
  QJsonArray toutesLesNotes;
  for (QString note : notes) toutesLesNotes.append(note.toDouble());
  json["notes"] = toutesLesNotes;
  QJsonDocument envoi(json);
  service.sendTextMessage(envoi.toJson());
}
```

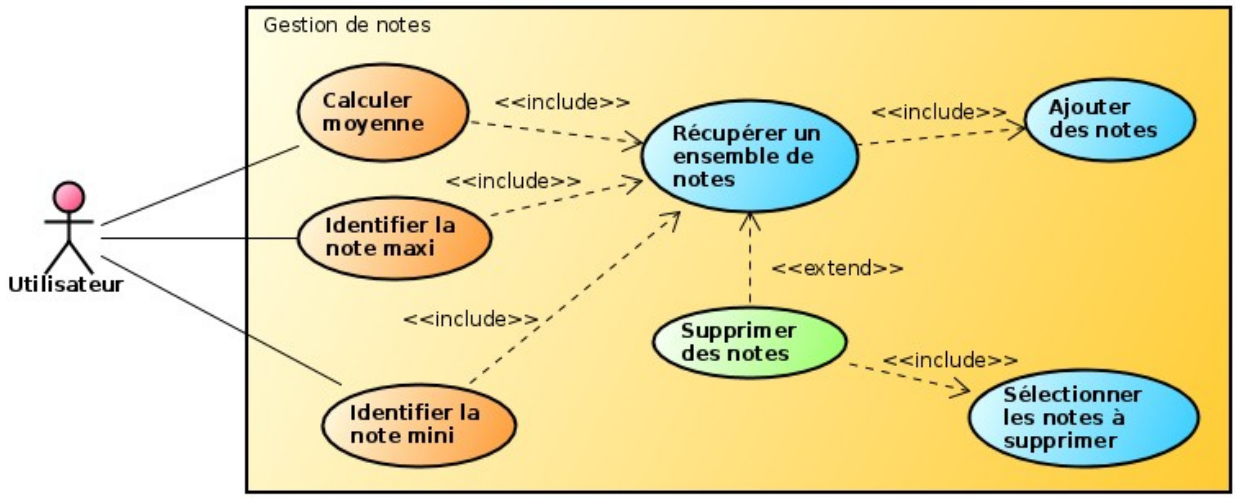


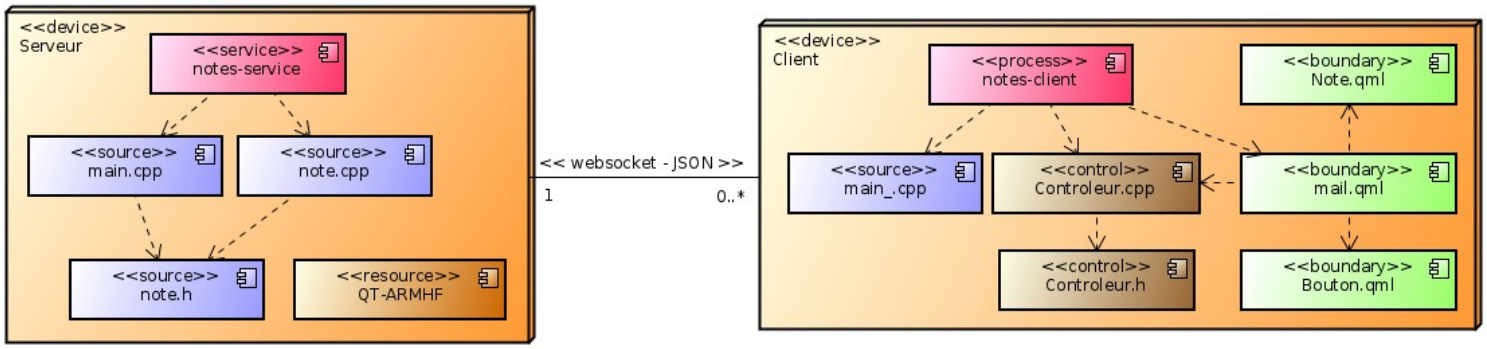
```qml
import QtQuick 2.0
import QtQuick.Controls 2.12

Button {
  anchors {
    left: parent.left
    leftMargin: 20
    right: parent.right
    rightMargin: 20
  }
}
```

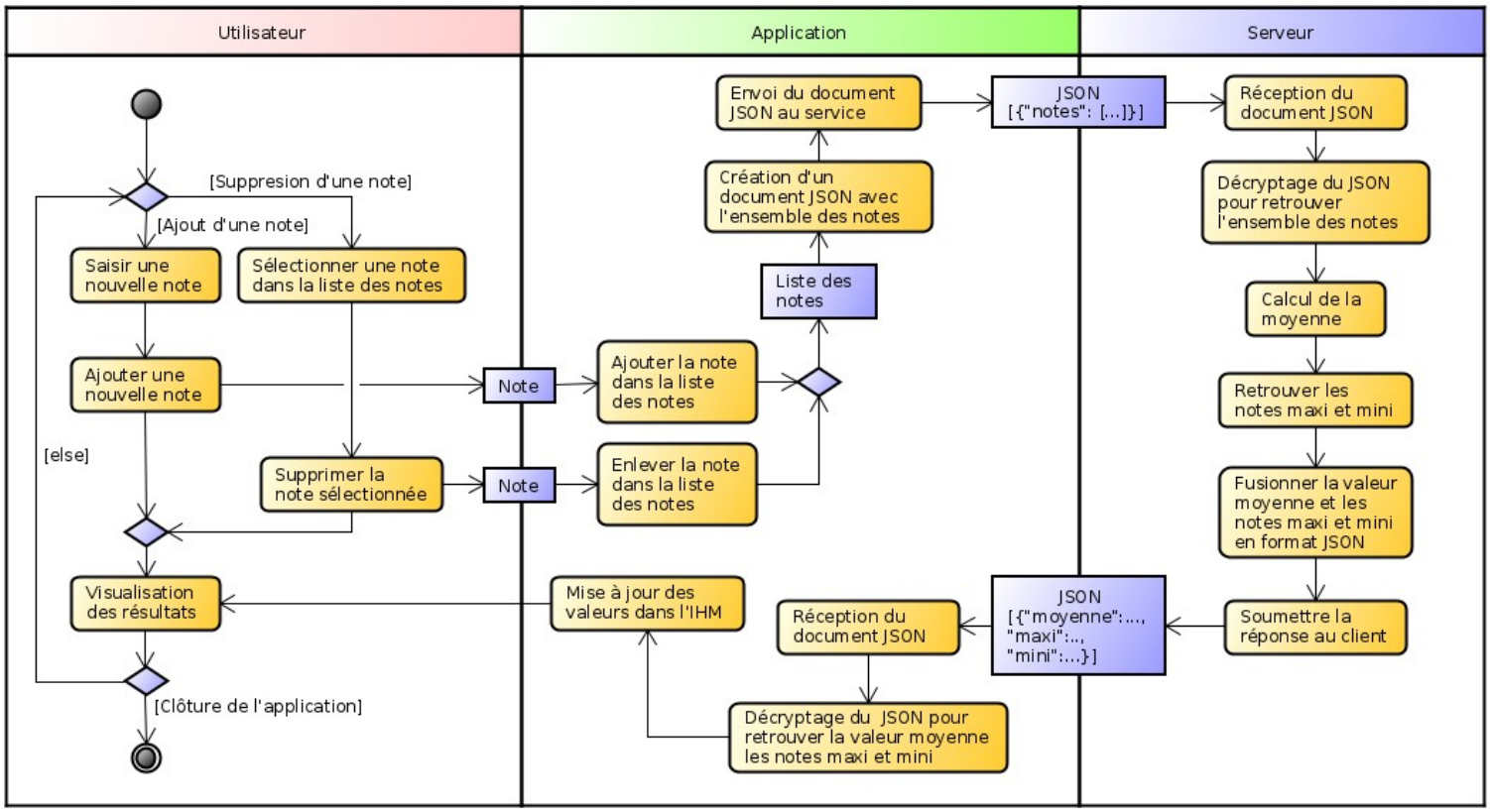


```qml
import QtQuick 2.0
import QtQuick.Controls 2.12

TextField {
  property string nom

  font.pointSize: 16
  font.bold: true
  rightPadding: 10
  horizontalAlignment: TextInput.AlignRight
  text: "0.0"
  width: 200
  readOnly: true

  anchors {
    left: parent.left
    leftMargin: 20
    right: parent.right
    rightMargin: 20
  }

  Text {
    text: nom
    color: "darkred"
    font.pointSize: 14
    font.italic: true
    anchors {
      verticalCenter: parent.verticalCenter
      left: parent.left
      leftMargin: 10
    }
  }
}
```

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtQuick.Controls 2.12

Window {
  visible: true
  width: 320
  height: 480
  color: "beige"
  title: qsTr("Gestion de notes")

  Column {
    anchors.fill: parent
    padding: 20
    spacing: 20

    Note {
      id: nouvelle
      nom: "Note"
      readOnly: false
    }

    Bouton {
      text: "Ajouter"
      onClicked: control.ajouterNote(nouvelle.text)
    }

    Bouton {
      text: "Supprimer"
      onClicked: control.supprimer(nouvelle.text)
    }

    ComboBox {
      id: liste
      anchors {
        left: parent.left
        leftMargin: 20
        right: parent.right
        rightMargin: 20
      }
      model: control.notes
      onCurrentTextChanged: nouvelle.text = currentText
    }

    Note { id: moyenne; nom: "Moyenne" }
    Note { id: maxi; nom: "Maxi"    }
    Note { id: mini; nom: "Mini"    }
  }

  Connections {
    target: control
    onValeursChanged: {
      moyenne.text = control.moyenne
      maxi.text = control.maxi
      mini.text = control.mini
      liste.currentIndex = liste.count-1
    }
  }
}
```


Gestion de notes
Utilisateur
Calculer moyenne
Identifier la note maxi
Identifier la note mini
Récupérer un ensemble de notes
Ajouter des notes
Supprimer des notes
Sélectionner les notes à supprimer
<<include>>
<<extend>>


<<device>>
Serveur
<<service>>
notes-service
<<source>>
main.cpp
<<source>>
note.cpp
<<source>>
note.h
<<resource>>
QT-ARMHF
<< websocket - JSON >>
1
0..*
<<device>>
Client
<<process>>
notes-client
<<boundary>>
Note.qml
<<source>>
main_.cpp
<<control>>
Controleur.cpp
<<boundary>>
mail.qml
<<control>>
Controleur.h
<<boundary>>
Bouton.qml


Utilisateur
Application
Serveur
[Suppresion d'une note]
[Ajout d'une note]
Saisir une nouvelle note
Sélectionner une note dans la liste des notes
Ajouter une nouvelle note
Note
Ajouter la note dans la liste des notes
[else]
Supprimer la note sélectionnée
Note
Enlever la note dans la liste des notes
Visualisation des résultats
[Clôture de l'application]
Envoi du document JSON au service
JSON [{"notes": [...]}]
Réception du document JSON
Création d'un document JSON avec l'ensemble des notes
Décryptage du JSON pour retrouver l'ensemble des notes
Liste des notes
Calcul de la moyenne
Retrouver les notes maxi et mini
Fusionner la valeur moyenne et les notes maxi et mini en format JSON
Mise à jour des valeurs dans l'IHM
Réception du document JSON
JSON [{"moyenne":..., "maxi":..., "mini":...}]
Soumettre la réponse au client
Décryptage du JSON pour retrouver la valeur moyenne les notes maxi et mini