За каждым клиентским приложением стоит система контроля и управления доступом. В этой статье описано создание приложения для авторизации пользователей с использованием Qt/C++. Создание приложения будет разбито на 4 этапа :

- Подготовка файлов приложения.

- Разработка диалоговых окон.

- Реализация классов.

- Инициализация БД и интерфейса к базе данных.

Возможно, Вам поможет изучение следующих статей этого ресурса:

<https://ravesli.com/urok-57-vvedenie-v-std-string/>

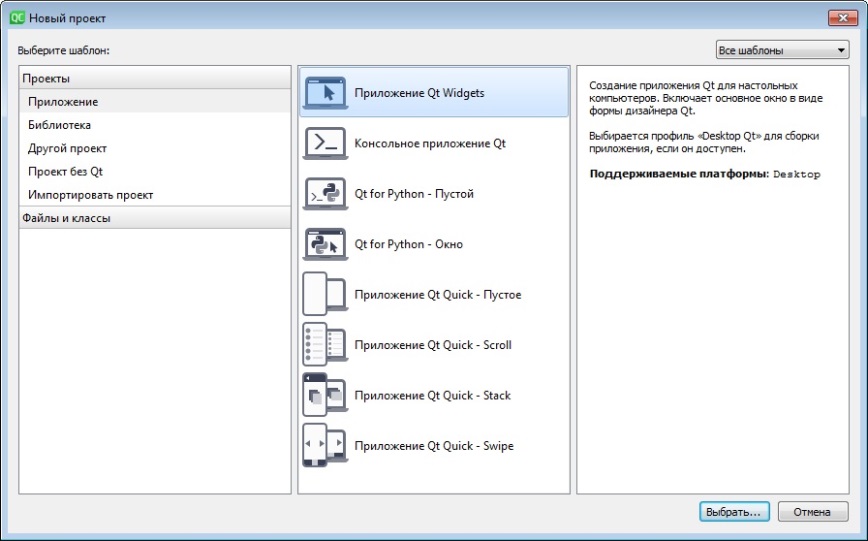
<https://ravesli.com/urok-147-kompozitsiya-obektov/>

<https://ravesli.com/urok-9-sloty-signaly-i-sobytiya-v-qt5/>

<https://ravesli.com/urok-13-sozdanie-polzovatelskogo-vidzheta-v-qt5/>

# Подготовка файлов приложения

Для создания этого приложения начните новый проект в Qt Creator. Для этого я воспользуюсь мастером создания новых проектов. В стартовом меню QtCreator я выбираю Новый проект>Приложение>Приложение QtWidgets.



Shot1

Новый проект будет создан, когда вы укажите его имя, директорию и выберете подходящий для него компилятор. В этом проекте уже будет заголовочный файл, форма и файл класса для главного окна MainWindow.

В этом приложении будет использована база данных, для регистрации, хранения и извлечения пользовательской информации. Для этого нам необходимо подключить модуль QtSQL в конфигурации проекта – файле, имеющем имя проекта и расширение .pro.



Shot2

Чтобы работать с классами этого модуля, потребуется включить заголовочный метафайл.

Классы модуля QtSQL разделяются на три уровня:

1. Уровень драйверов.

2.Программный уровень.

3.Уровень пользовательского интерфейса.

К первому уровню относятся классы для получения данных на физическом уровне.

Классы второго уровня предоставляют программный интерфейс для обращения к базе данных.

Третий уровень предоставляет модели для отображения результатов запросов в представлениях интервью.

Это приложение будет использовать классы второго уровня. Таким образом можно будет рассмотреть применение объектов класса QString для управления базой данных.

Qt поддерживает следующие системы управления базами данных(СУБД):

|  |  |
| --- | --- |
| Идентификатор | Описание |
| QOCI | БД Oracle v7,8,9 |
| QODBC | ODBC-сервер для Microsoft SQL Server, IBM DB2, Sybase SQL, iODBC и некоторых других |
| QMYSQL | [www.mysql.com](http://www.mysql.com) |
| QTDS | Sybase Adaptive Server |
| QPSQL | БД PostgreSQL с поддержкой SQL92/SQL3 |
| QSQLITE | SQLite v2 |
| QSQLITE | SQLite v3+ |
| QIBASE | Borland InterBase |
| QDB2 | DB2 от IBM |

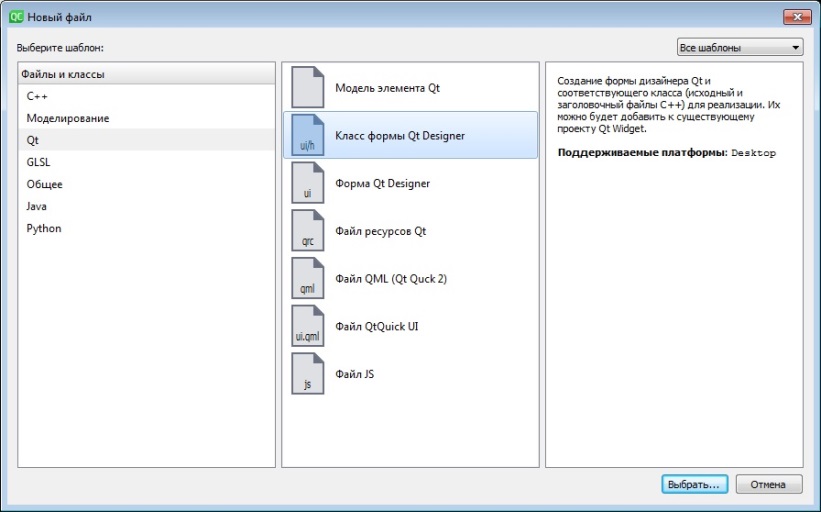
Вы можете использовать идентификатор из этой таблицы для подключения к базе данных. Если Вы не видите здесь нужную вам СУБД – вам придется написать для нее свой драйвер.

Это приложение будет хранить свои данные локально, поэтому я воспользуюсь SQLite, так как драйвер этой СУБД и сама база по умолчанию всегда распространяются вместе с Qt.  
Это приложение в своей работе будет оперировать тремя окнами – mainwindow, auth\_window и reg\_window, как главным окном, окном авторизации и окном регистрации пользователя.

В проекте, созданном как Приложение QtWidgets по умолчанию создается окно MainWindow ассоциированное с классом MainWindow.

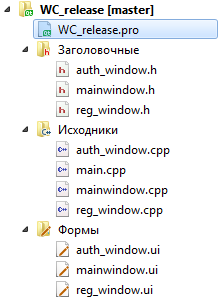
Для реализации окон авторизации и регистрации добавим к проекту два новых класса формы QtDesigner.

Для этого вызовите контекстное меню от папки проекта в обозревателе проектов, выберите Добавить новый.  
Далее Qt>Класс формы Qt Designer>выбрать.



Shot4

Теперь в нашем проекте есть все нужные файлы, и необходимый модуль для работы с базой данных.



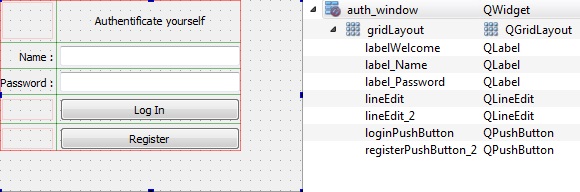
Shot3

# Разработка диалоговых окон

Перед тем как реализовывать классы можно разработать внешний вид диалоговых окон. Закончив этот этап возвращаться к нему больше не придется.

Идея этого приложения такова – положение и размер главного окна запоминаются для каждого авторизованного пользователя. Авторизация и регистрация происходят в отдельных окнах, принадлежащих главному окну. Соединение с базой данных также принадлежит главному окну.

Начнем с разработки окна авторизации. Для этого откройте файл auth\_window.ui, QtCreator сразу же откроет его в редакторе.



Shot5

Для начала подберите размер окна, растягивая его за синие маркеры расположенные по контуру. Затем разместите слой позиционирования – Layout. Просто перетащите его из панели инструментов в ваше окно. Для того, чтобы скомпоновать виджеты окна как на рисунке лучше всего подойдет gridLayout. Впрочем, Вы можете справиться и без него.

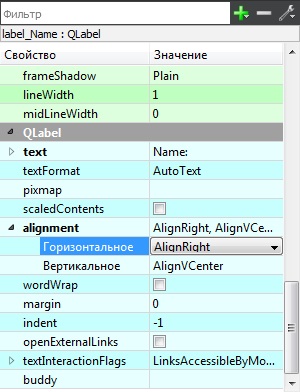
Далее можно разместить виджеты окна – кнопки, поля, ярлыки. Перетаскивайте их с панели инструментов в слой позиционирования. Каждый новый виджет будет перераспределять положение всех виджетов слоя. Как Вы можете видеть – в этом окне используются виджеты QLabel, QLineEdit и QPushButton(не считая QGridLayout). Закончив размещать виджеты в окне воспользуйтесь инструментом изменения порядка обхода(табуляции), кнопка для включения этого инструмента находится на панели сразу над областью редактирования.

Shot6



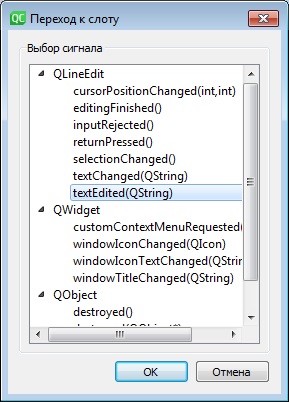
Рекомендую дать виджетам говорящие названия. Название loginPushButton явно говорит о назначении этого объекта, в отличии от lineEdit\_2.

Вы можете изменить атрибуты виджета, как собственные, так и унаследованные в области атрибутов. Например можно разместить надписи виджетов Qlabel по правому краю изменив свойство QLabel.alignment.



Shot9

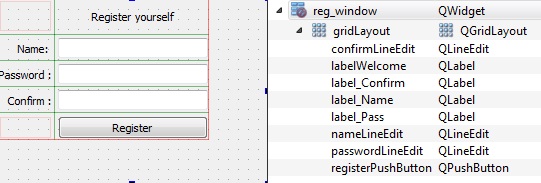
Последний этап проектирования окна – объявление слотов. Для этого вызовите контекстное меню виджета и выберите пункт Перейти к слоту. В появившемся окне выберите нужный слот.



Shot7

Для работы этого приложения нам понадобятся слоты textEdited(QString) созданные от виджетов QLineEdit   
и clicked() созданные от виджетов QPushButton. Прототипы слотов будут созданы в заголовочном файле, а их реализация появится в файле класса auth\_window.cpp.

Разработка внешнего вида окна авторизации закончена. Перейдем к разработке внешнего вида окна регистрации. Для откройте файл reg\_window.ui.



Shot8

Повторите шаги сделанные для предыдущего окна: подберите размер окна, разместите слой позиционирования, разместите виджеты в слое, проименуйте созданные виджеты, создайте слоты textEdited(QString) для полей   
и clicked() для кнопки.  
Не забудьте расставить порядок табуляции.

# Реализация классов

Все файлы и необходимые им слоты созданы, теперь можно приступить к описанию классов.

Начинать реализовывать классы лучше с объявления членов класса в заголовочном файле.

Классу MainWindow для работы понадобятся заголовочные файлы библиотек QString, QMainWindow и QtSql.

mainwindow.h

#include <QMainWindow>

#include <QString>

#include <QtSql/QtSql>

#include "auth\_window.h"

#include "reg\_window.h"

Также необходимо подключить заголовочные файлы окон регистрации и авторизации.

namespace **Ui** {

class **MainWindow**;

}

Хотя QtCreator поместил MainWindow по умолчанию в пространство имен Ui – не все функции класса будут сосредоточены на пользовательском интерфейсе. В более сложных приложениях следует строже следовать стратегии именования.

Так как я намерен вызывать главное окно лишь после успешной авторизации, мне понадобится создать публичный метод отображения главного – display().

Для работы с базой данных мне понадобятся публичный метод connectDB()а также экзмепляр подключения к БД mw\_db и строка для формирования запроса к БД db\_input.

Использование прочих членов класса можно будет проследить в файле класса.

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

explicit **MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

void **display**(); //прототип пользовательской функции отображения

bool **connectDB**(); //прототип метода подключения к БД

private:

Ui::MainWindow \*ui\_Main;

auth\_window ui\_Auth; //экземпляры окна авторизации и окна регистрации

reg\_window ui\_Reg; //принадлежат главному окну

QString m\_username; //строки для обработки

QString m\_userpass; //пользовательского ввода

QString db\_input; //строка для отправки запроса к БД

QSqlDatabase mw\_db; //экземпляр подключения к БД

int user\_counter; //счетчик пользователей

bool m\_loginSuccesfull; //флаг успешной авторизации

private slots:

void **authorizeUser**(); //пользовательские слоты

void **registerWindowShow**();

void **registerUser**();

};

auth\_window.h

Кроме автоматически созданных членов и слотов, классу окна авторизации понадобятся сигналы, которые окно авторизации будет посылать главному окну. Также понадобится пара переменных для хранения пользовательского ввода и геттеры для их получения.

#include <QWidget>

namespace **Ui** {

class **auth\_window**;

}

class **auth\_window** : public QWidget

{

Q\_OBJECT

public:

explicit **auth\_window**(QWidget \*parent = nullptr);

~***auth\_window***();

QString **getLogin**();

QString **getPass**();

signals:

void **login\_button\_clicked**();

void **register\_button\_clicked**();

private slots:

void **on\_lineEdit\_textEdited**(const QString &arg1);

void **on\_lineEdit\_2\_textEdited**(const QString &arg1);

void **on\_loginPushButton\_clicked**();

void **on\_registerPushButton\_2\_clicked**();

private:

Ui::auth\_window \*ui;

QString m\_username;

QString m\_userpass;

};

reg\_window.h

Окну регистрации понадобится сигнал, который оно будет посылать главному окну, переменные для хранения пользовательского ввода и геттеры для них, а также я решил добавить функцию проверки пользователя на внимательность.

#include <QWidget>

namespace **Ui** {

class **reg\_window**;

}

class **reg\_window** : public QWidget

{

Q\_OBJECT

public:

explicit **reg\_window**(QWidget \*parent = nullptr);

~***reg\_window***();

QString **getName**();

QString **getPass**();

bool **checkPass**();

signals:

void **register\_button\_clicked2**();

private slots:

void **on\_nameLineEdit\_textEdited**(const QString &arg1);

void **on\_passwordLineEdit\_textEdited**(const QString &arg1);

void **on\_confirmLineEdit\_textEdited**(const QString &arg1);

void **on\_registerPushButton\_clicked**();

private:

Ui::reg\_window \*ui;

QString m\_userName;

QString m\_userPass;

QString m\_confirmation;

};

auth\_window.cpp

В автоматически созданных файлах класса содержимое конструкторов создается по умолчанию. В этом классе меня это устраивает.

#include "auth\_window.h"

#include "ui\_auth\_window.h"

auth\_window::**auth\_window**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::auth\_window)

{

ui->setupUi(this);

}

auth\_window::~***auth\_window***()

{

delete ui;

}

В автоматически созданных слотах кнопок размещаю сигналы, заявленные в заголовочном файле

void auth\_window::**on\_registerPushButton\_2\_clicked**()

{

emit register\_button\_clicked();

}

void auth\_window::**on\_loginPushButton\_clicked**()

{

emit login\_button\_clicked();

}

В автоматически созданных слотах полей копирую содержимое полей в переменные класса.

void auth\_window::**on\_lineEdit\_textEdited**(const QString &arg1)

{

auth\_window::m\_username = arg1;

}

void auth\_window::**on\_lineEdit\_2\_textEdited**(const QString &arg1)

{

auth\_window::m\_userpass = arg1;

}

Таким образом геттеры возвращают содержимое полей.

QString auth\_window::**getLogin**()

{

return auth\_window::m\_username;

}

QString auth\_window::**getPass**()

{

return auth\_window::m\_userpass;

}

Reg\_window.cpp

Конструктор и деструктор по умолчанию

reg\_window::**reg\_window**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::reg\_window)

{

ui->setupUi(this);

}

reg\_window::~***reg\_window***()

{

delete ui;

}

Слот кнопки регистрации вызывает сигнал для главного окна

void reg\_window::**on\_registerPushButton\_clicked**()

{

emit register\_button\_clicked2();

}

Слоты полей копируют пользовательский ввод

void reg\_window::**on\_nameLineEdit\_textEdited**(const QString &arg1)

{

reg\_window::m\_userName = arg1;

}

void reg\_window::**on\_passwordLineEdit\_textEdited**(const QString &arg1)

{

reg\_window::m\_userPass = arg1;

}

void reg\_window::**on\_confirmLineEdit\_textEdited**(const QString &arg1)

{

reg\_window::m\_confirmation = arg1;

}

Геттеры возвращают пользовательский ввод

QString reg\_window::**getName**()

{

return m\_userName;

}

QString reg\_window::**getPass**()

{

return m\_userPass;

}

Функция checkPass возвращает true, если пользователь не ошибся при подтверждении пароля в поле confirm

bool reg\_window::**checkPass**()

{

return (m\_confirmation == m\_userPass);

}

# Инициализация БД и интерфейса к БД

main\_window.cpp

До сих пор все было несложно, и каждый член класса содержал не больше одного выражения. Вся реализация проявится именно сейчас, в файле класса главного окна.

Первым делом подключим файлы прочих классов с помощью директивы include.

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "auth\_window.h"

#include "reg\_window.h"

#include <QtDebug>

Модуль QtDebug пригодится для отображения ошибок в программе в окно QtCreator, вместо отображения в консоль или другой виджет.

В конструкторе главного окна можно соединить сигналы со слотами и осуществить соединение с БД, а также создать в ней новую таблицу. Библиотека QtSql обеспечивает нам доступ к сообщениям с ошибками, которые формируются запросами QSqlQuery.

MainWindow::**MainWindow**(QWidget \*parent) :   
MainWindow(parent),

ui\_Main(new Ui::MainWindow)

{

user\_counter = 0;

m\_loginSuccesfull = false;

connect(&ui\_Auth, SIGNAL(login\_button\_clicked()),

this, SLOT(authorizeUser()));

connect(&ui\_Auth,SIGNAL(destroyed()),

this, SLOT(show()));

connect(&ui\_Auth,SIGNAL(register\_button\_clicked()),

this,SLOT(registerWindowShow()));

connect(&ui\_Reg,SIGNAL(register\_button\_clicked2()),

this,SLOT(registerUser()));

connect(&ui\_Reg,SIGNAL(destroyed()),

&ui\_Auth, SLOT(show()));

if(!connectDB())

{

qDebug() << "Failed to connect DB";

}

QSqlQuery query;

db\_input = "CREATE TABLE userlist ( "

"number INTEGER PRIMARY KEY NOT NULL,"

"name VARCHAR(20), "

"pass VARCHAR(12), "

"xpos INTEGER, "

"ypos INTEGER, "

"width INTEGER, "

"length INTEGER );";

if(!query.exec(db\_input))

{

qDebug() << "Unable to create a table" << query.lastError();

}

ui\_Main->setupUi(this);

}

Функция authorizeUser() соединена с сигналом кнопки авторизации, она сохраняет пользовательский ввод, составляет из него запрос к БД, проверяет соответствие хранимого пароля введенному пользователем и, в случае успеха – открывает MainWindow с параметрами, сохраненными в предыдущую сессию.

void MainWindow::**authorizeUser**()

{

m\_username = ui\_Auth.getLogin();

m\_userpass = ui\_Auth.getPass();

QString str\_t = " SELECT \* "

" FROM userlist "

" WHERE name = '%1'";

//int db\_number = 0;

QString username = "";

QString userpass = "";

int xPos = 0;

int yPos = 0;

int width = 0;

int length = 0;

db\_input = str\_t.arg(m\_username);

QSqlQuery query;

QSqlRecord rec;

if(!query.exec(db\_input))

{

qDebug() << "Unable to execute query - exiting" << query.lastError() << " : " << query.lastQuery();

}

rec = query.record();

query.next();

user\_counter = query.value(rec.indexOf("number")).toInt();

username = query.value(rec.indexOf("name")).toString();

userpass = query.value(rec.indexOf("pass")).toString();

if(m\_username != username || m\_userpass != userpass)

{

qDebug() << "Password missmatch" << username << " " << userpass;

m\_loginSuccesfull = false;

}

else

{

m\_loginSuccesfull = true;

xPos = query.value(rec.indexOf("xpos")).toInt();

yPos = query.value(rec.indexOf("ypos")).toInt();

width = query.value(rec.indexOf("width")).toInt();

length = query.value(rec.indexOf("length")).toInt();

ui\_Auth.close();

ui\_Reg.close();

this->setGeometry(xPos,yPos,width, length);

this->show();

}

}

Функция registerWindowShow() соединена с сигналом register\_button\_clicked(). Она скрывает окно авторизации и вызывает окно регистрации.

void MainWindow::**registerWindowShow**()

{

ui\_Auth.hide();

ui\_Reg.show();

}Функция registerUser соединена с сигналом кнопки регистрации, принадлежащей окну регистрации. Она выполняет проверку пользовательского ввода, получает новый номер записи из БД и регистрирует пользователя. Вы могли заметить, что у поля name при создании новой БД не было идентификатора unique, остальная программа также не выполняет проверки на уникальность регистрируемого имени.

void MainWindow::**registerUser**()

{

if(ui\_Reg.checkPass())

{

QSqlQuery query;

QSqlRecord rec;

QString str\_t = "SELECT COUNT(\*) "

"FROM userlist;";

db\_input = str\_t;

if(!query.exec(db\_input))

{

qDebug() << "Unable to get number " << query.lastError() << " : " << query.lastQuery();

return;

}

else

{

query.next();

rec = query.record();

user\_counter = rec.value(0).toInt();

qDebug() << user\_counter;

}

m\_username = ui\_Reg.getName();

m\_userpass = ui\_Reg.getPass();

user\_counter++;

str\_t = "INSERT INTO userlist(number, name, pass, xpos, ypos, width, length)"

"VALUES(%1, '%2', '%3', %4, %5, %6, %7);";

db\_input = str\_t .arg(user\_counter)

.arg(m\_username)

.arg(m\_userpass)

.arg(0)

.arg(0)

.arg(800)

.arg(400);

if(!query.exec(db\_input))

{

qDebug() << "Unable to insert data" << query.lastError() << " : " << query.lastQuery();

}

else

{

ui\_Reg.hide();

ui\_Auth.show();

}

}

else

{

qDebug() << "Confirm password coorectly";

}

}

Метод display() подменяет вызов главного окна вызовом окна авторизации

void MainWindow::**display**()

{

ui\_Auth.show();

}

Деструктор главного окна, в случае удачной авторизации обновляет содержимое записи в БД и прерывает соединение с БД. Затем уничтожает окна приложения.

MainWindow::~***MainWindow***()

{

if(m\_loginSuccesfull)

{

QString str\_t = "UPDATE userlist "

"SET xpos = %2, ypos = %3, width = %4, length = %5 "

"WHERE name = '%1';";

db\_input = str\_t .arg(m\_username)

.arg(this->x())

.arg(this->y())

.arg(this->width())

.arg(this->height());

QSqlQuery query;

if(!query.exec(db\_input))

{

qDebug() << "Unable to insert data" << query.lastError() << " : " << query.lastQuery() ;

}

}

mw\_db.removeDatabase("authorisation");

qDebug() << "MainWindow Destroyed";

delete ui\_Main;

exit(0);

}

Функция connectDB() создает соединение с БД и устанавливает её имя. После того, как эта функция будет выполнена впервые, в директории программы появится файл с именем, указанным здесь для базы данных.

bool MainWindow::**connectDB**()

{

mw\_db = QSqlDatabase::addDatabase("QSQLITE");

mw\_db.setDatabaseName("authorisation");

if(!mw\_db.open())

{

qDebug() << "Cannot open database: " << mw\_db.lastError();

return false;

}

return true;

}

# Заключение

Теперь если Вы решите написать приложение, требующее авторизации пользователей – Вы знаете, как это может быть организовано. В Вашем приложении параметрами авторизации могут быть голос, отпечатки пальцев, фото пользователя или что-то еще. Вы наверняка будете передавать пользовательский ввод для авторизации на удаленный сервер, обеспечите уникальность каждого имени и зашифруете канал связи. Уверен, у Вас все получится.