stable.h:

#ifndef STABLE\_H

#define STABLE\_H

// Precompiled headers

// QT Libs

#include <QApplication>

#include <QDialog>

#include <QMainWindow>

#include <QMessageBox>

#include <QSettings>

#include <QSqlDatabase>

#include <QSqlError>

#include <QSqlQuery>

#include <QString>

#include <QStyle>

#include <QtGlobal>

#include <QDate>

#include <QSqlRelationalTableModel>

#include <QObject>

#include <QSqlRecord>

#endif // STABLE\_H

database.h

#ifndef DATABASE\_H

#define DATABASE\_H

#include <user.h>

#include "Singleton.h"

#include "stable.h"

class Database : public Singleton<Database> {

public:

Database();

bool createConnection();

void disconect();

bool createTableUsers();

bool createTableLicenses();

bool createTableProgram();

bool createTableLicensor();

bool createTableComputer();

bool login(QString, QString);

void addUser(User);

QSqlQuery getQueryEndedLics();

QString lastError();

private:

QSqlDatabase db;

int port;

QString hostname;

QString dbname;

QString username;

QString password;

};

#endif // DATABASE\_H

database.cpp:

#include "database.h"

Database::Database()

: Singleton<Database>(\*this)

{

QSettings config("database.ini", QSettings::IniFormat);

config.beginGroup("db");

this->hostname = config.value("hostname").toString();

this->dbname = config.value("db\_name").toString();

this->password = config.value("password").toString();

this->port = config.value("port").toInt();

this->username = config.value("user").toString();

}

bool Database::createConnection()

{

db = QSqlDatabase::addDatabase("QPSQL");

db.setHostName(this->hostname);

db.setPort(this->port);

db.setUserName(this->username);

db.setPassword(this->password);

db.open();

QSqlQuery query(db);

query.exec("CREATE DATABASE " + this->dbname);

db.close();

db.setDatabaseName(this->dbname);

if (db.open()) {

createTableUsers();

createTableComputer();

createTableLicensor();

createTableProgram();

createTableLicenses();

} else

return false;

return true;

}

void Database::disconect()

{

if (db.isOpen())

db.close();

}

bool Database::createTableUsers()

{

QSqlQuery query(db);

bool res = query.exec(QString("CREATE TABLE IF NOT EXISTS USERS(") + QString("ID SERIAL PRIMARY KEY,") + QString("FNAME VARCHAR NOT NULL,") + QString("LNAME VARCHAR NOT NULL, ") + QString("LOGIN VARCHAR NOT NULL, ") + QString("PASSWORD VARCHAR NOT NULL, ") + QString("UROLE VARCHAR NOT NULL)"));

qDebug() << query.lastError().text();

return res;

}

bool Database::createTableLicenses()

{

QSqlQuery query(db);

bool res = query.exec(QString("CREATE TABLE IF NOT EXISTS LICENSES(") + QString("ID SERIAL PRIMARY KEY,") + QString("START\_L DATE NOT NULL,") + QString("END\_L DATE NOT NULL,") + QString("PRICE INTEGER NOT NULL,") + QString("NAME VARCHAR NOT NULL, ") +

QString("COMPUTER\_ID INTEGER NOT NULL, FOREIGN KEY (COMPUTER\_ID) REFERENCES COMPUTERS(ID) ON DELETE CASCADE, ") +

QString("PROGRAM\_ID INTEGER NOT NULL, FOREIGN KEY (PROGRAM\_ID) REFERENCES PROGRAMS(ID) ON DELETE CASCADE, ") +

QString("LICENSOR\_ID INTEGER NOT NULL, FOREIGN KEY (LICENSOR\_ID) REFERENCES LICENSORS(ID) ON DELETE CASCADE)"));

qDebug() << query.lastError().text();

return res;

}

bool Database::createTableProgram()

{

QSqlQuery query(db);

bool res = query.exec(QString("CREATE TABLE IF NOT EXISTS PROGRAMS(") + QString("ID SERIAL PRIMARY KEY,") + QString("NAME VARCHAR NOT NULL)") );

qDebug() << query.lastError().text();

return res;

}

bool Database::createTableLicensor()

{

QSqlQuery query(db);

bool res = query.exec(QString("CREATE TABLE IF NOT EXISTS LICENSORS(") + QString("ID SERIAL PRIMARY KEY ,") + QString("ADDRESS VARCHAR NOT NULL,") + QString("NAME VARCHAR NOT NULL)"));

qDebug() << query.lastError().text();

return res;

}

bool Database::createTableComputer()

{

QSqlQuery query(db);

bool res = query.exec(QString("CREATE TABLE IF NOT EXISTS COMPUTERS(") + QString("ID SERIAL PRIMARY KEY,") + QString("NAME VARCHAR NOT NULL)"));

qDebug() << query.lastError().text();

return res;

}

bool Database::login(QString login, QString password)

{

QSqlQuery query(db);

query.prepare("SELECT \* FROM USERS WHERE LOGIN = :login AND PASSWORD = :password");

query.bindValue(":login", login);

query.bindValue(":password", password);

query.exec();

query.first();

User user(query.value(0).toInt(), query.value(1).toString(),query.value(2).toString(), query.value(3).toString(), query.value(5).toString());

User::setAuth(user);

return User::getAuth().login != "";

}

void Database::addUser(User u)

{

QSqlQuery query(db);

query.prepare("INSERT INTO USERS(LNAME, FNAME, LOGIN, PASSWORD, UROLE) VALUES (:ln, :fn, :login, :password, :role)");

query.bindValue(":fn", u.fname);

query.bindValue(":ln", u.lname);

query.bindValue(":login", u.login);

query.bindValue(":password", u.password);

query.bindValue(":role", u.role);

query.exec();

qDebug() << query.lastError();

}

QSqlQuery Database::getQueryEndedLics()

{

return QSqlQuery(QString("SELECT LIC.\* FROM LICENSES LIC WHERE END\_L < NOW()"));

}

QString Database::lastError()

{

return db.lastError().text();

}

DbManager.h:

#ifndef DBMANAGER\_H

#define DBMANAGER\_H

#include "Singleton.h"

#include "stable.h"

#include <user.h>

class DbManager : public Singleton<DbManager> {

public:

DbManager();

bool connect();

bool login(QString, QString);

void addUser(User);

};

#endif // DBMANAGER\_H

DbManager.cpp:

#include "DbManager.h"

#include "database.h"

DbManager::DbManager()

: Singleton<DbManager>(\*this)

{

}

bool DbManager::connect()

{

//подключение к бд

return Database::instance().createConnection();

}

bool DbManager::login(QString login, QString password)

{

return Database::instance().login(login, password);

}

void DbManager::addUser(User u)

{

Database::instance().addUser(u);

}

main.cpp:

#include "mainwindow.h"

#include "stable.h"

int main(int argc, char\* argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

addnewdialog.h:

#ifndef ADDNEWDIALOG\_H

#define ADDNEWDIALOG\_H

#include <stable.h>

#include <user.h>

namespace Ui {

class AddNewDialog;

}

class AddNewDialog : public QDialog {

Q\_OBJECT

public:

explicit AddNewDialog(QWidget\* parent = nullptr);

~AddNewDialog();

User data;

private slots:

void on\_buttonBox\_accepted();

private:

Ui::AddNewDialog\* ui;

};

#endif // ADDNEWDIALOG\_H

addnewdialog.cpp:

#include "addnewdialog.h"

#include "ui\_addnew.h"

AddNewDialog::AddNewDialog(QWidget\* parent)

: QDialog(parent)

, ui(new Ui::AddNewDialog)

{

ui->setupUi(this);

}

AddNewDialog::~AddNewDialog()

{

delete ui;

}

void AddNewDialog::on\_buttonBox\_accepted()

{

data.fname = ui->fname->text();

data.lname = ui->lname->text();

data.login = ui->login->text();

data.password = ui->password->text();

data.role = ui->role->currentText();

this->accept();

}

commonwindow.h:

#ifndef COMMONWINDOW\_H

#define COMMONWINDOW\_H

#include "stable.h"

#include <QMainWindow>

namespace Ui {

class CommonWindow;

}

class CommonWindow : public QMainWindow {

Q\_OBJECT

public:

explicit CommonWindow(QWidget\* parent = nullptr, const QString title = "Window", const QString table = "licenses");

~CommonWindow();

private slots:

void about();

void logout();

void on\_users\_triggered();

void on\_licensors\_triggered();

void on\_pos\_triggered();

void on\_pks\_triggered();

void on\_addBtn\_clicked();

void on\_detBtn\_clicked();

void on\_updateTable\_clicked();

void on\_action\_triggered();

private:

Ui::CommonWindow\* ui;

QString tablename;

QSqlRelationalTableModel \*model;

void setHeaders();

void setRelations();

};

#endif // COMMONWINDOW\_H

commonwindow.cpp:

#include "commonwindow.h"

#include "ui\_commonwindow.h"

#include <mainwindow.h>

#include <querydialog.h>

CommonWindow::CommonWindow(QWidget\* parent, const QString title, const QString table)

: QMainWindow(parent)

, ui(new Ui::CommonWindow)

{

ui->setupUi(this);

if(User::getAuth().role != "Администратор"){

ui->addBtn->setEnabled(false);

ui->detBtn->setEnabled(false);

}

qDebug() << User::getAuth().login + " " << User::getAuth().role;

if(table == "users") ui->addBtn->setEnabled(false);

this->tablename = table;

this->setWindowTitle(title);

ui->logout->addAction("Выход", this, SLOT(logout()));

ui->about->addAction("О программе", this, SLOT(about()));

this->model = new QSqlRelationalTableModel(this);

this->model->setTable(tablename);

setRelations();

this->ui->tableView->setModel(model);

setHeaders();

ui->tableView->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->tableView->horizontalHeader()->setSectionResizeMode(0, QHeaderView::Fixed);

ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);

ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

if(User::getAuth().role != "Администратор"){

model->setEditStrategy(QSqlRelationalTableModel::EditStrategy::OnManualSubmit);

ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);

} else model->setEditStrategy(QSqlRelationalTableModel::OnFieldChange);

model->select();

ui-> tableView->setSortingEnabled(true);

}

CommonWindow::~CommonWindow()

{

delete ui;

}

void CommonWindow::about()

{

QMessageBox::about(this, "О программе", "Программа создана для учета лицензий программного обеспечения.");

}

void CommonWindow::logout()

{

this->parentWidget()->show();

this->close();

}

void CommonWindow::setHeaders()

{

QStringList headers;

if(this->tablename == "licenses")

headers << tr("ID") << tr("Начало") << tr("Окончание")<< tr("Цена")<< tr("Название")<< tr("Рабочая станция")<< tr("ПО")<< tr("Орг.");

else if(this->tablename == "users") {

headers << tr("ID") << tr("Имя") << tr("Фамилия") << tr("Логин") << tr("Пароль") << tr("Роль");

this->ui->tableView->setColumnHidden(4, true);

} else if (this->tablename == "programs")

headers << tr("ID") << tr("Название");

else if (this->tablename == "licensors")

headers << tr("ID") << tr("Адрес") << tr("Наименование");

else if (this->tablename == "computers")

headers << tr("ID") << tr("Название");

for(int i = 0, j = 0; i < model->columnCount(); i++, j++)

model->setHeaderData(i,Qt::Horizontal,headers[j]);

}

void CommonWindow::setRelations()

{

if(this->tablename == "licenses"){

model->setRelation(5, QSqlRelation("computers", "id", "name"));

model->setRelation(6, QSqlRelation("programs", "id", "name"));

model->setRelation(7, QSqlRelation("licensors", "id", "name"));

}

}

void CommonWindow::on\_users\_triggered()

{

CommonWindow\* w = new CommonWindow(this, "Пользователи", "users");

w->setAttribute(Qt::WA\_DeleteOnClose, true);

w->show();

}

void CommonWindow::on\_licensors\_triggered()

{

CommonWindow\* w = new CommonWindow(this, "Организации", "licensors");

w->setAttribute(Qt::WA\_DeleteOnClose, true);

w->show();

}

void CommonWindow::on\_pos\_triggered()

{

CommonWindow\* w = new CommonWindow(this, "Программное обеспечение", "programs");

w->setAttribute(Qt::WA\_DeleteOnClose, true);

w->show();

}

void CommonWindow::on\_pks\_triggered()

{

CommonWindow\* w = new CommonWindow(this, "Рабочие станции", "computers");

w->setAttribute(Qt::WA\_DeleteOnClose, true);

w->show();

}

void CommonWindow::on\_addBtn\_clicked()

{

this->model->insertRow(model->rowCount());

}

void CommonWindow::on\_detBtn\_clicked()

{

this->model->removeRow(ui->tableView->selectionModel()->selectedRows().at(0).row());

model->select();

}

void CommonWindow::on\_updateTable\_clicked()

{

this->model->select();

}

void CommonWindow::on\_action\_triggered()

{

QueryDialog\* w = new QueryDialog(1, this);

w->setAttribute(Qt::WA\_DeleteOnClose, true);

w->show();

}

mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "stable.h"

#include "addnewdialog.h"

#include "commonwindow.h"

#include "database.h"

#define dbManager DbManager::instance()

QT\_BEGIN\_NAMESPACE

namespace Ui {

class MainWindow;

}

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow {

Q\_OBJECT

public:

MainWindow(QWidget\* parent = nullptr);

~MainWindow();

private slots:

void on\_login\_clicked();

void on\_registration\_clicked();

void on\_close\_Dialog();

void on\_destroyed\_child();

private:

Ui::MainWindow\* ui;

Database& db = Database::instance();

AddNewDialog\* dialog;

CommonWindow\* main;

QString login;

QString password;

};

#endif // MAINWINDOW\_H

mainwindow.cpp:

#include "DbManager.h"

#include "addnewdialog.h"

#include "ui\_mainwindow.h"

#include <mainwindow.h>

MainWindow::MainWindow(QWidget\* parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

//подключение к бд

if (!dbManager.connect()) {

QMessageBox::warning(this, "Ошибка", db.lastError());

exit(1);

}

}

MainWindow::~MainWindow()

{

delete ui;

delete dialog;

}

//вход

void MainWindow::on\_login\_clicked()

{

if (dbManager.login(ui->login\_->text(), ui->password->text())) {

main = new CommonWindow(this, "Лицензии ПО");

main->setAttribute(Qt::WA\_DeleteOnClose, true);

main->show();

ui->login\_->setText("");

ui->password->setText("");

QObject::connect(main, SIGNAL(destroyed()) ,this, SLOT(on\_destroyed\_child()));

this->hide();

} else

QMessageBox::warning(this, "Ошибка", "Неправильный логин или пароль.");

ui->login\_->setText("");

ui->password->setText("");

}

//регистрация

void MainWindow::on\_registration\_clicked()

{

dialog = new AddNewDialog(this);

QObject::connect(dialog, SIGNAL(accepted()), this, SLOT(on\_close\_Dialog()));

dialog->exec();

}

void MainWindow::on\_close\_Dialog()

{

if (dialog->data.fname != "" && dialog->data.lname != "" && dialog->data.login != "" && dialog->data.password != "" && dialog->data.role != "") {

User usr = dialog->data;

dbManager.addUser(usr);

}

}

void MainWindow::on\_destroyed\_child()

{

qDebug() << "i'm died!";

exit(0);

}

querydialog.h:

#ifndef QUERYDIALOG\_H

#define QUERYDIALOG\_H

#include "stable.h"

#include "database.h"

namespace Ui {

class QueryDialog;

}

class QueryDialog : public QDialog

{

Q\_OBJECT

public:

explicit QueryDialog(int state, QWidget \*parent = nullptr);

~QueryDialog();

private:

Ui::QueryDialog \*ui;

QSqlRelationalTableModel \*model;

};

#endif // QUERYDIALOG\_H

querydialog.cpp:

#include "querydialog.h"

#include "ui\_querydialog.h"

QueryDialog::QueryDialog(int state, QWidget \*parent) :

QDialog(parent),

ui(new Ui::QueryDialog)

{

ui->setupUi(this);

QStringList headers;

model = new QSqlRelationalTableModel(this);

ui->tableView->setModel(model);

switch(state){

case 1:

model->setQuery(Database::instance().getQueryEndedLics());

headers << tr("ID") << tr("Начало") << tr("Окончание")<< tr("Цена")<< tr("Название")<< tr("Рабочая станция")<< tr("ПО")<< tr("Орг.");

model->setRelation(5, QSqlRelation("computers", "id", "name"));

model->setRelation(6, QSqlRelation("programs", "id", "name"));

model->setRelation(7, QSqlRelation("licensors", "id", "name"));

for(int i = 0, j = 0; i < model->columnCount(); i++, j++)

model->setHeaderData(i,Qt::Horizontal,headers[j]);

model->select();

}

ui->tableView->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

ui->tableView->horizontalHeader()->setSectionResizeMode(0, QHeaderView::ResizeToContents);

ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection);

ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows);

model->setEditStrategy(QSqlRelationalTableModel::EditStrategy::OnManualSubmit);

ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);

ui-> tableView->setSortingEnabled(true);

}

QueryDialog::~QueryDialog()

{

delete ui;

}

base.h:

#ifndef BASE\_H

#define BASE\_H

#include <stable.h>

class Base {

public:

Base();

long id;

};

#endif // BASE\_H

base.cpp:

#include "../../inc/model/base.h"

Base::Base()

{

}

computer.h:

#ifndef COMPUTER\_H

#define COMPUTER\_H

#include "base.h"

class Computer : public Base {

public:

Computer();

QString name;

};

#endif // COMPUTER\_H

computer.cpp:

#include "../../inc/model/computer.h"

Computer::Computer()

{

}

license.h:

#ifndef LICENSE\_H

#define LICENSE\_H

#include "base.h"

class License : public Base {

public:

License();

QDate start;

QDate end;

long computer;

long licensor;

long program;

};

#endif // LICENSE\_H

license.cpp:

#include "model/license.h"

License::License()

{

}

licensor.h:

#ifndef LICENSOR\_H

#define LICENSOR\_H

#include <base.h>

class Licensor : public Base {

public:

Licensor();

QString name;

QString address;

};

#endif // LICENSOR\_H

licensor.cpp:

#include "../../inc/model/licensor.h"

Licensor::Licensor()

{

}

program.h:

#ifndef PROGRAM\_H

#define PROGRAM\_H

#include <base.h>

class Program : public Base

{

public:

Program();

QString name;

};

#endif // PROGRAM\_H

program.cpp:

#include "../../inc/model/program.h"

Program::Program()

{

}

user.h:

#ifndef USER\_H

#define USER\_H

#include "stable.h"

#include <base.h>

class User : public Base {

public:

User() { }

User(long id, QString fn, QString ln, QString login, QString role);

QString fname;

QString lname;

QString login;

QString password;

QString role;

virtual ~User() { }

static User getAuth();

static void setAuth(const User newAuth);

private:

static User auth;

};

#endif // USER\_H

user.cpp:

#include "user.h"

User::User(long id, QString fn, QString ln, QString login, QString role)

{

this->id = id;

fname = fn;

lname = ln;

this-> login = login;

this-> role = role;

}

User User::getAuth()

{

return auth;

}

void User::setAuth(const User newAuth)

{

auth = newAuth;

}

User User::auth = User();